The African School of Fundamental Physics and Applications

Integrating Scientific Computing into Math and Science Classes

Dave Biersach
Brookhaven National Laboratory
dbiersach@bnl.gov

Session 03
Computing in Science

Brookhaven National Laboratory

Session **03** – Topics

- Learn about Python's **dictionary** data structure and how to read data files in **CSV** and **JSON** formats
- Analyze chemical trends and anomalies across the **Periodic Table of Elements**
- Implement the **linear regression** formulas to find the **line of best fit** using the method of least squares created by Gauss
- Identify an *unknown element* using the **Ideal gas law**
- Determine the height and velocity of **cosmic ray** showers using Newtonian Kinematics
- Simulate the **trajectory** of a circus cannon performer to determine a safe initial launch velocity

Run python_dictionaries.ipynb – **Cells 1...2**

Import the pretty printer module from the `pprint` package

```
[1]  # Cell 1
     from pprint import pprint
```

①

Create a dictionary (key/value pairs) of nations and their capitals

1. The keys will be of type `string`, and only one key will exist for each value
2. The values will be of type `string`, and only one value will exist for each key

However, Python dictionaries can have multiple items with a key or value
But you can never have repeated keys - all keys must be unique!

```
[2]  # Cell 2
     capitals = {
         "Morocco": "Rabat",
         "Germany": "Berlin",
         "France": "Paris",
         "South Africa": "Pretoria",
         "Russia": "Moscow",
         "Egypt": "Cairo",
         "USA": "Washington D.C.",
         "India": "New Delhi",
         "China": "Beijing",
     }
```

②

Run python_dictionaries.ipynb – **Cells 3...4**

"Pretty" print the `capitals` dictionary

Notice the pretty printer displays the dictionary sorted by keys

```
[3]  # Cell 3 ← ①
     pprint(capitals)
```

```
{'China': 'Beijing',
 'Egypt': 'Cairo',          ← ②
 'France': 'Paris',
 'Germany': 'Berlin',
 'India': 'New Delhi',
 'Morocco': 'Rabat',
 'Russia': 'Moscow',
 'South Africa': 'Pretoria',
 'USA': 'Washington D.C.'}
```

Read the value of a specific key

```
[4]  # Cell 4
     print(capitals["USA"]) ← ③
```

```
Washington D.C. ← ④
```

Run python_dictionaries.ipynb – **Cells 5...6**

Create/insert new key/value pairs into the dictionary using two different methods/syntax

```
[5]  # Cell 5
     capitals.update({"United Kingdom": "London"})
     capitals["Spain"] = "Madrid"
     pprint(capitals)
```

①

```
{'China': 'Beijing',
 'Egypt': 'Cairo',
 'France': 'Paris',
 'Germany': 'Berlin',
 'India': 'New Delhi',
 'Morocco': 'Rabat',
 'Russia': 'Moscow',
 'South Africa': 'Pretoria',
 'Spain': 'Madrid',
 'USA': 'Washington D.C.',
 'United Kingdom': 'London'}
```

②

Update/change/edit the value for a key that already exists in the dictionary

```
     # Cell 6
     capitals["USA"] = "New York"
     pprint(capitals)
```

③

```
{'China': 'Beijing',
 'Egypt': 'Cairo',
 'France': 'Paris',
 'Germany': 'Berlin',
 'India': 'New Delhi',
 'Morocco': 'Rabat',
 'Russia': 'Moscow',
 'South Africa': 'Pretoria',
 'Spain': 'Madrid',
 'USA': 'New York',
 'United Kingdom': 'London'}
```

④

Run python_dictionaries.ipynb – **Cell 7**

Delete/remove a key from the dictionary

```
[7]  # Cell 7
     del capitals["Russia"]
     pprint(capitals)
```

①

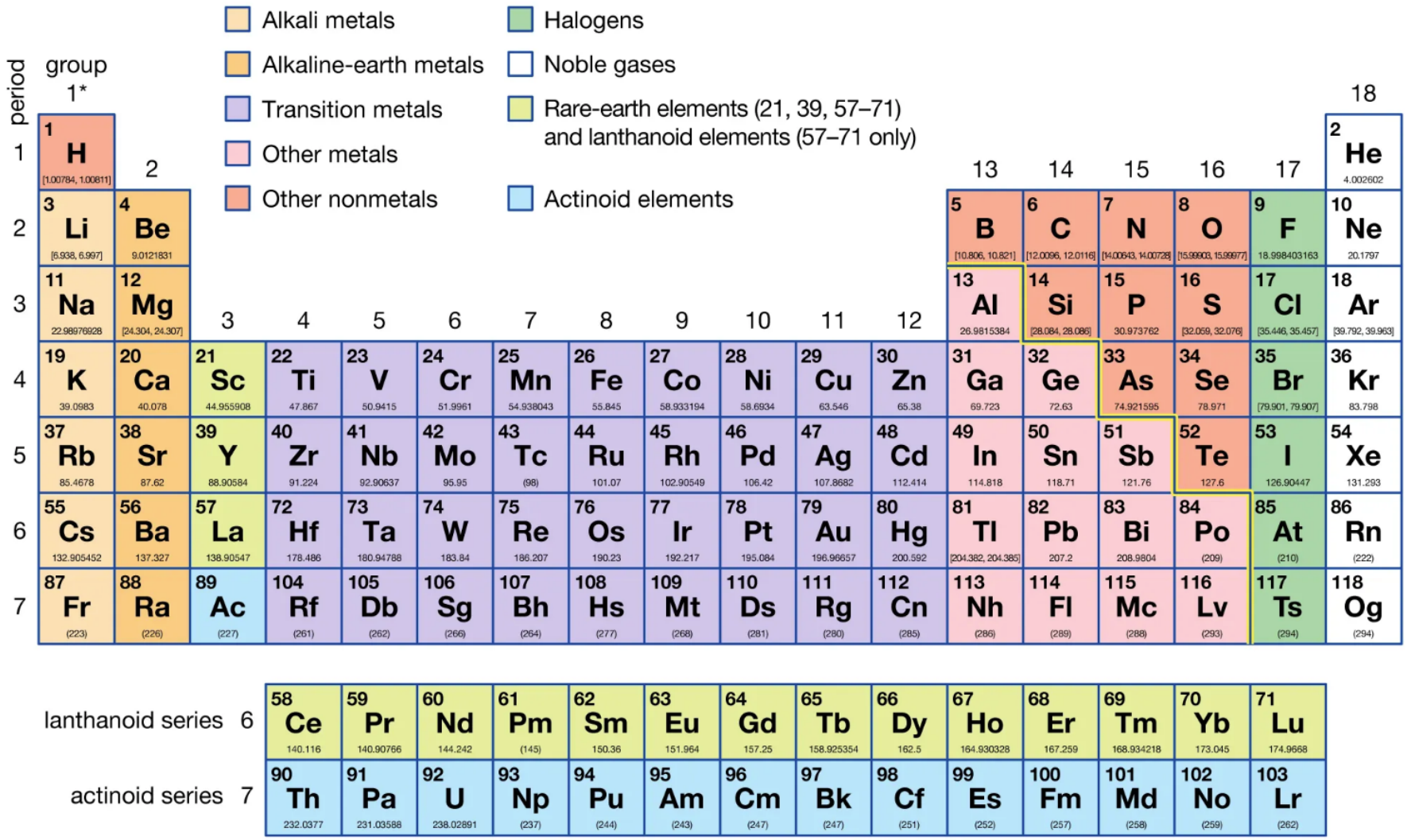
```
{'China': 'Beijing',
 'Egypt': 'Cairo',
 'France': 'Paris',
 'Germany': 'Berlin',
 'India': 'New Delhi',
 'Morocco': 'Rabat',
 'South Africa': 'Pretoria',
 'Spain': 'Madrid',
 'USA': 'New York',
 'United Kingdom': 'London'}
```

②

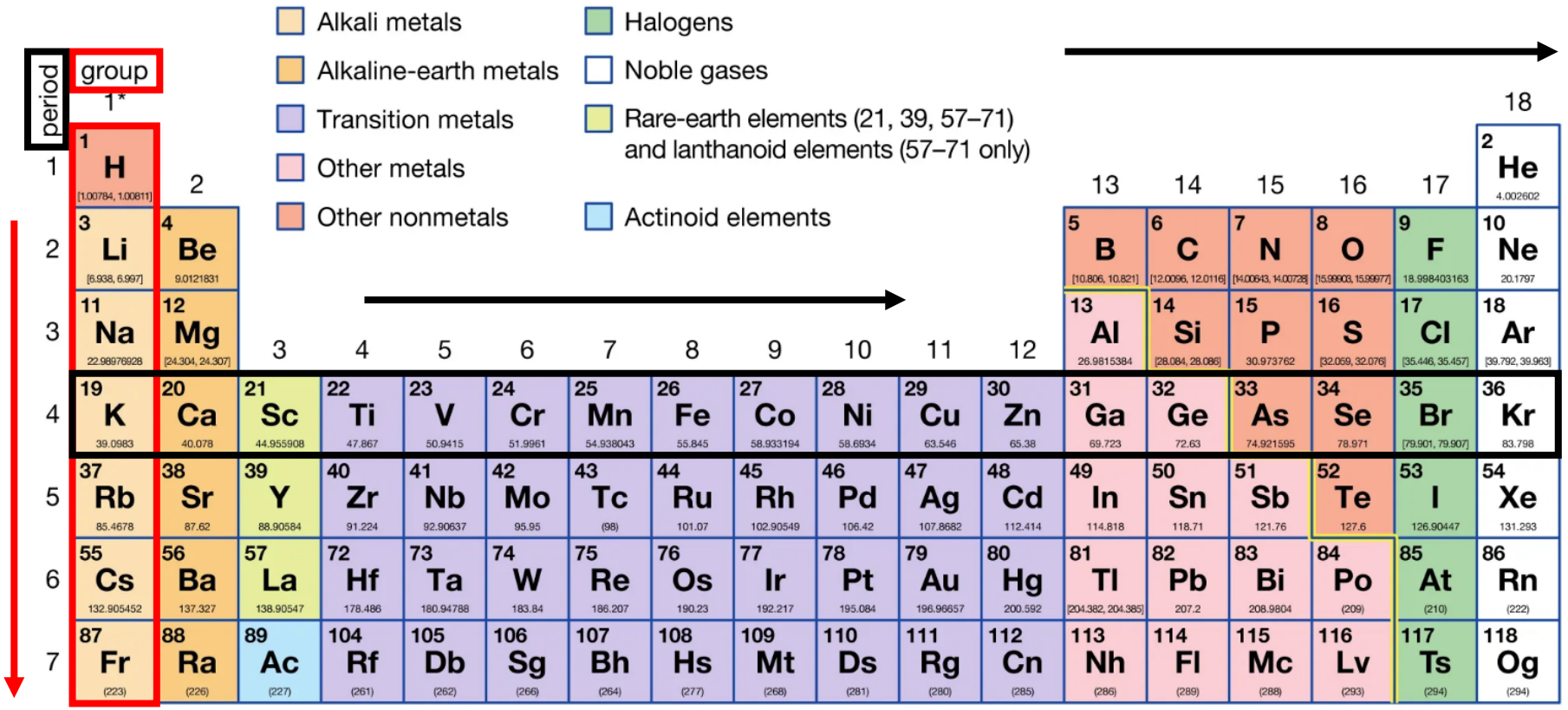
Liquid Range

- Your scientist has asked you to plot the **melting** and **boiling** point of each element across the periodic chart
- She is interested in visualizing any **trends** and *discontinuities* in elements having the same valence shell structure
- Therefore, she wants you to plot the elements first **by group**, then **by period**, then **by atomic number** order
- She has asked you to include elements from both the **Lanthanoid** and **Actinoid** series
- Is there a readily available **data set** on the web that encodes the entire periodic chart?

The Periodic Table (circa 2021)



The Periodic Table (circa 2021)



All elements in a **group** have the same number of valence *electrons*. As a result, elements in the same group often display similar properties and *reactivity*.

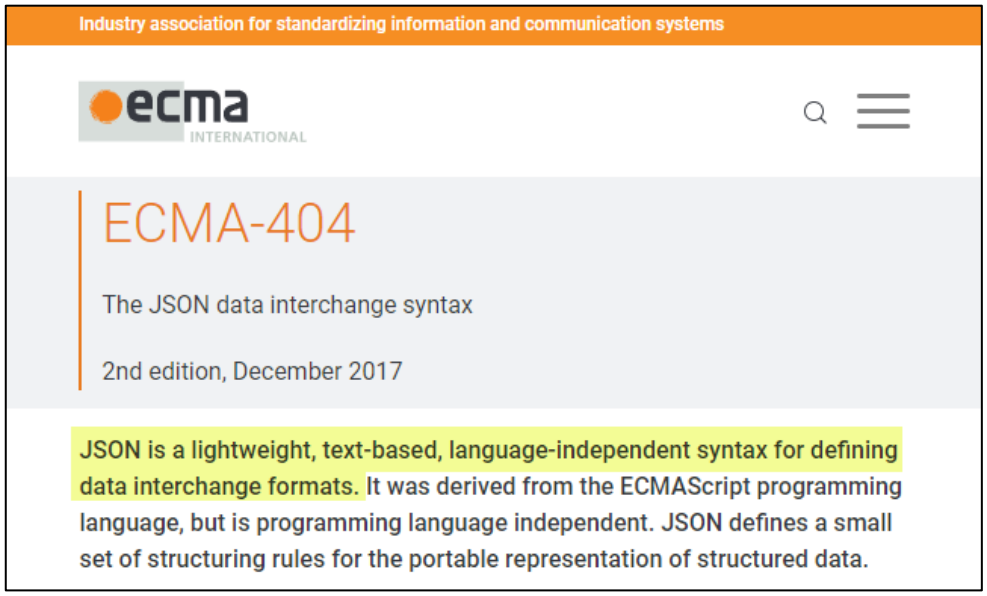
All elements in a **period** have the same number of electron shells. Each *next* element in a period **has one more proton** and is *less* metallic than its predecessor.

Python Dictionaries and JSON Data Files

Java**S**cript **O**bject **N**otation (JSON)

"Key-Value Pairs"

https://www.json.org

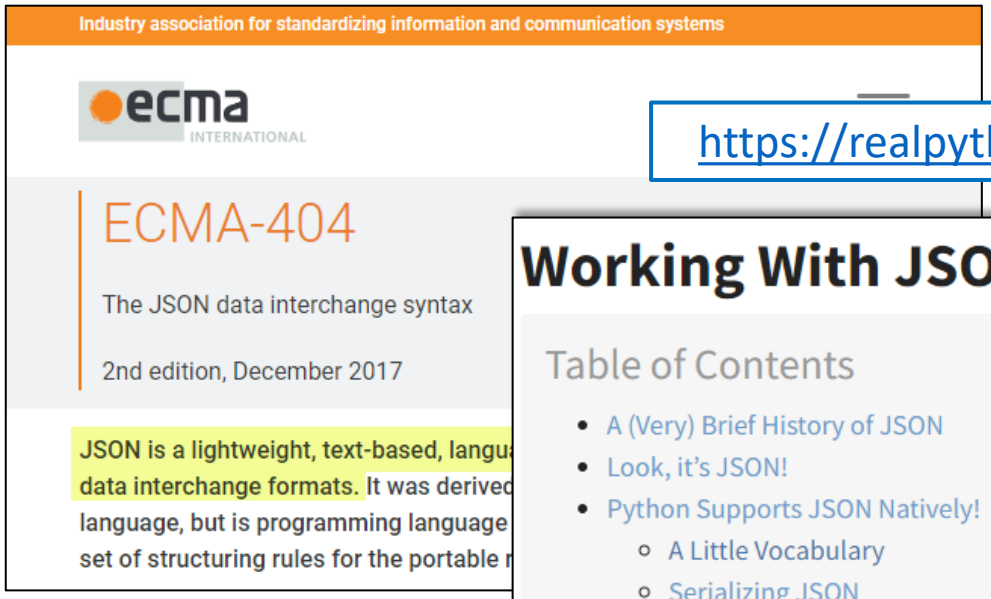


Python Dictionaries and JSON Data Files

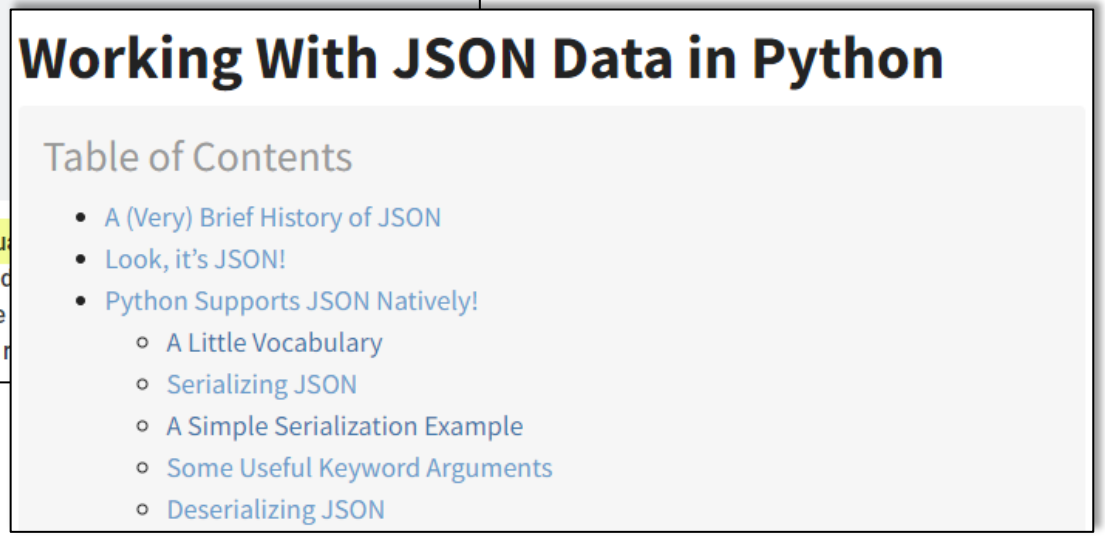
Java**S**cript **O**bject **N**otation (JSON)

"Key-Value Pairs"

https://www.json.org

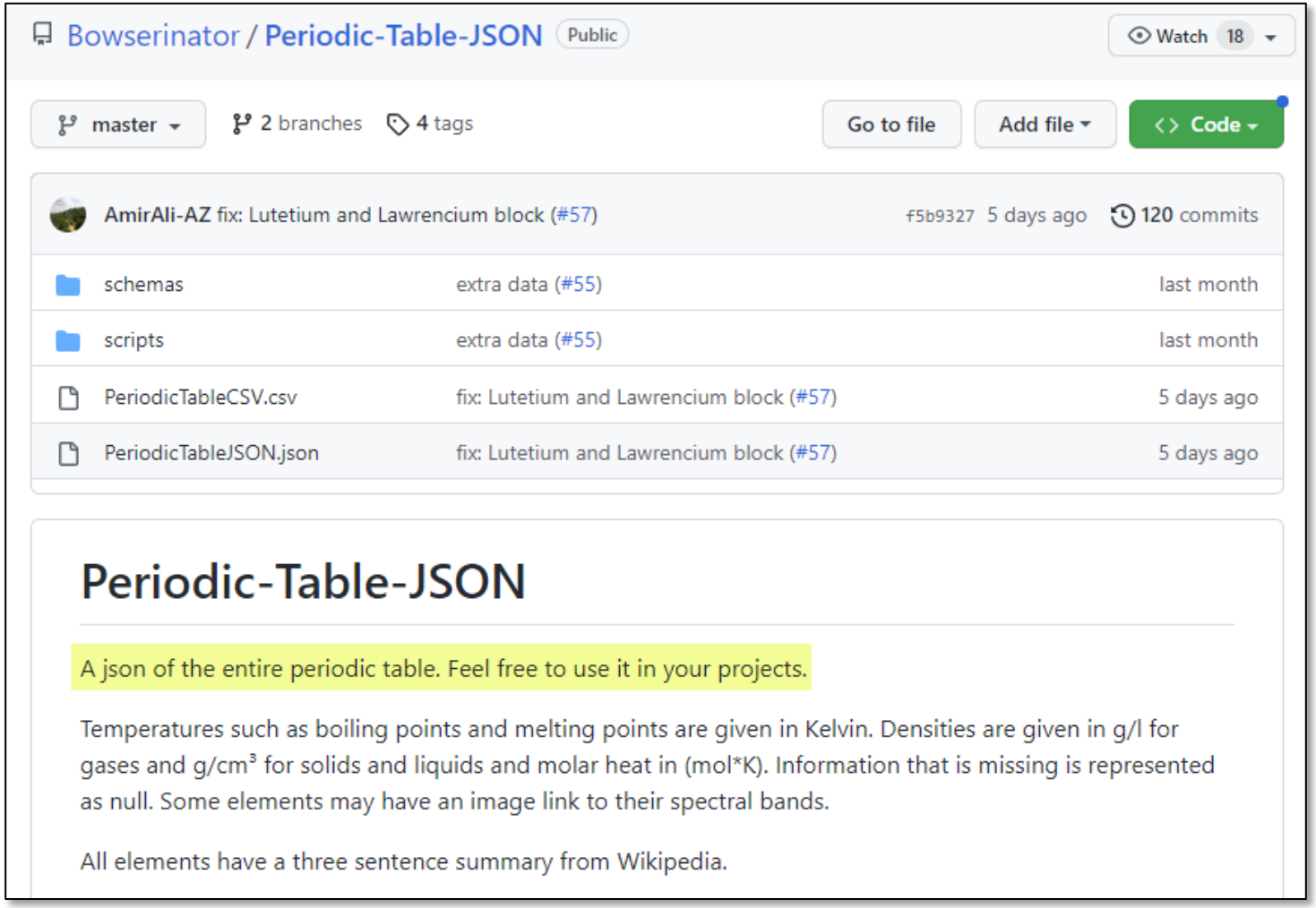


https://realpython.com/python-json



The Periodic Table as JSON

https://github.com/Bowserinator/Periodic-Table-JSON



periodic_table.json


```
{
  "name": "Hydrogen",
  "appearance": "colorless gas",
  "atomic_mass": 1.008,
  "boil": 20.271,
  "category": "diatomic nonmetal",
  "density": 0.08988,
  "discovered_by": "Henry Cavendish",
  "melt": 13.99,
  "molar_heat": 28.836,
  "named_by": "Antoine Lavoisier",
  "number": 1,
  "period": 1,
  "group": 1,
  "phase": "Gas",
  "source": "https://en.wikipedia.org/wiki/Hydrogen",
  "bohr_model_image": "https://storage.googleapis.com/search-ar-edu/periodic-table/element_001_hydrogen/
  "bohr_model_3d": "https://storage.googleapis.com/search-ar-edu/periodic-table/element_001_hydrogen/ele
  "spectral_img": "https://en.wikipedia.org/wiki/File:Hydrogen_Spectra.jpg",
  "summary": "Hydrogen is a chemical element with chemical symbol H and atomic number 1. With an atomic
  "symbol": "H",
  "xpos": 1,
  "ypos": 1,
  "wxpos": 1,
  "wypos": 1,
  "shells": [
    1
  ],
  "electron_configuration": "1s1",
  "electron_configuration_semantic": "1s1",
  "electron_affinity": 72.769,
  "electronegativity_pauling": 2.2,
  "ionization_energies": [
    1312
  ],
  "cpk-hex": "ffffff",
  "image": {
    "title": "Vial of glowing ultrapure hydrogen, H2. Original size in cm: 1 x 5",
    "url": "https://upload.wikimedia.org/wikipedia/commons/d/d9/Hydrogenglow.jpg",
    "attribution": "User:Jurii, CC BY 3.0 <https://creativecommons.org/licenses/by/3.0>, via Wikimedia C
  },
  "block": "s"
}
```


⑥ ⑤ ② ④ ③ ①

JSON is like a Python dictionary with keys and *values*

A value can itself be a "nested" dictionary

Run plot_liquid_range.ipynb – **Cells 1...2**

Import necessary packages/modules

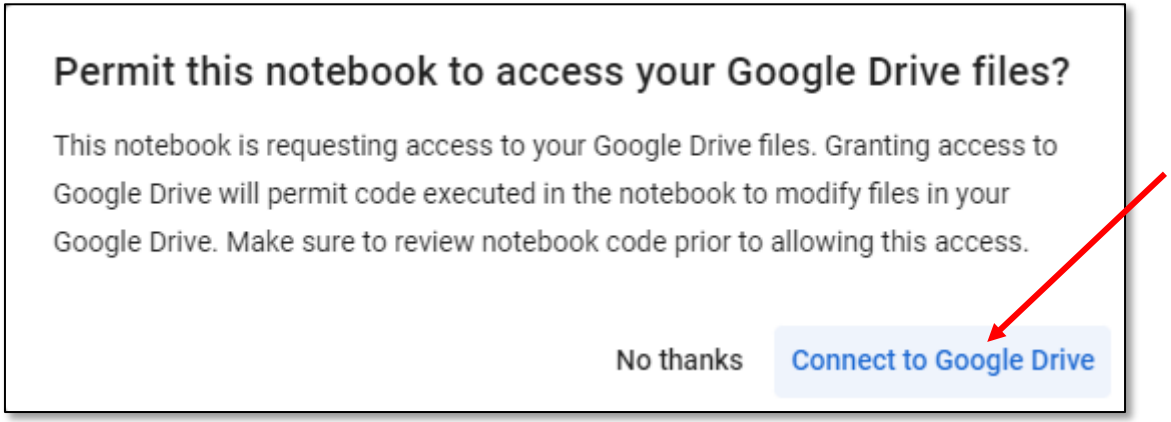
```
[1]  # Cell 1
     import json                          ①
     from pathlib import Path             ②

     import matplotlib.pyplot as plt
     import numpy as np
     from google.colab import drive       ③

     import pandas as pd                  ④
```

Connect this notebook to your Google Drive

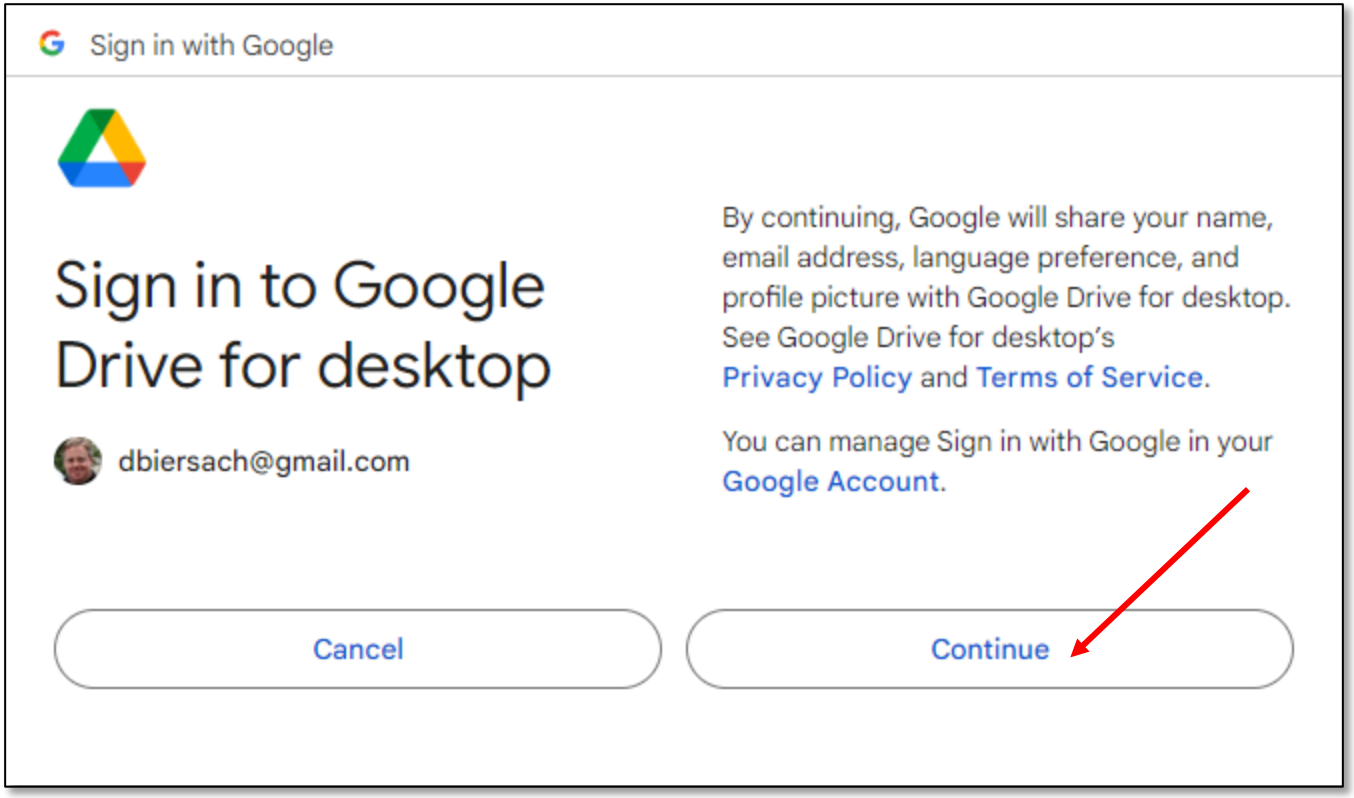
```
[2]  # Cell 2
     drive.mount("/content/gdrive", force_remount=True)                    ⑤
     notebook_path = Path("/content/gdrive/MyDrive/asp-hs")
     notebook_path = notebook_path / Path("Session 03 - Computing in Science")   ⑥
     notebook_path
```

```
Mounted at /content/gdrive
PosixPath('/content/gdrive/MyDrive/asp-hs/Session 03 - Computing in Science')   ⑦
```

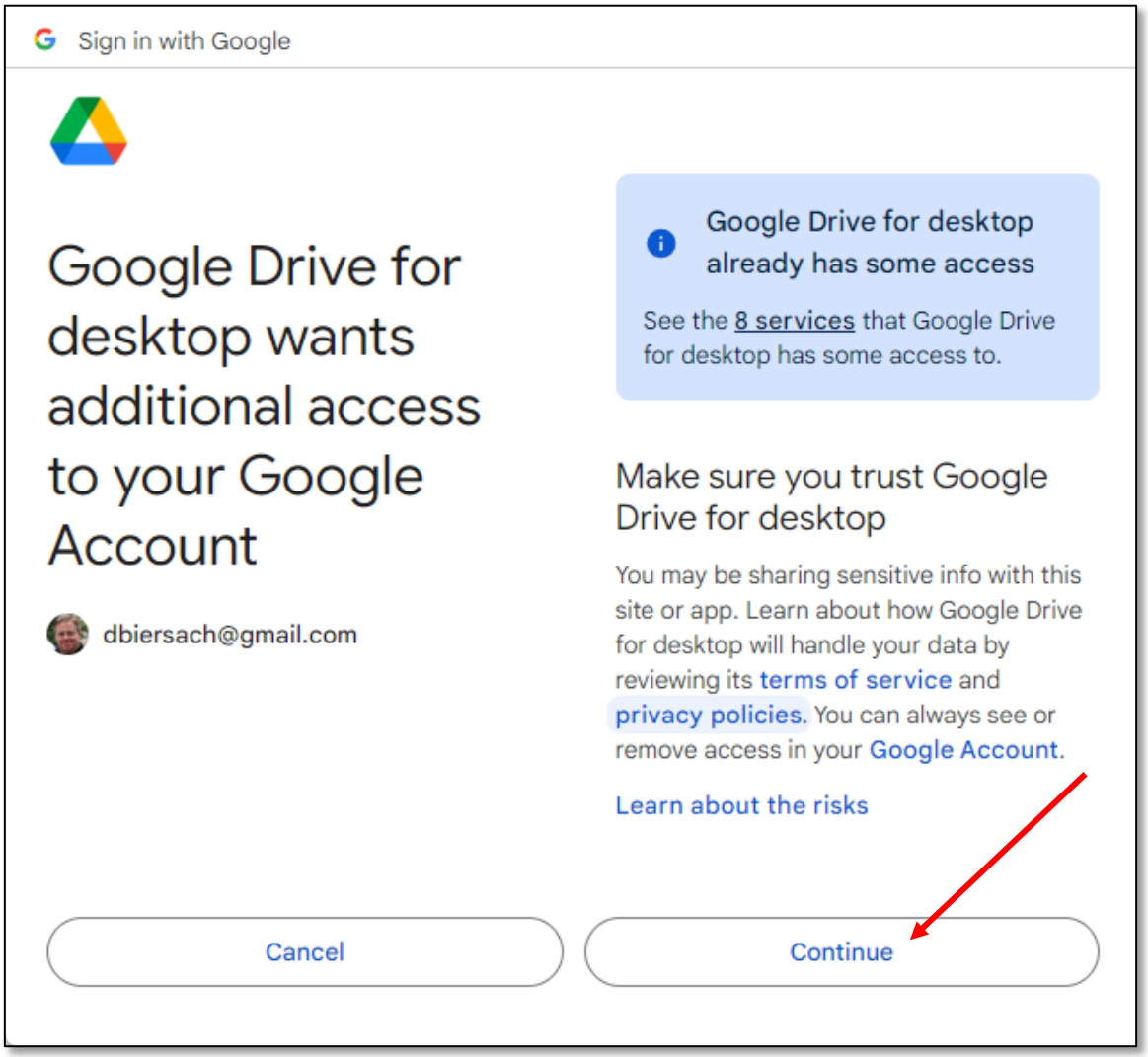
View plot_liquid_range.ipynb – **Cell 2**



View plot_liquid_range.ipynb – **Cell 2**



View plot_liquid_range.ipynb – **Cell 2**



View plot_liquid_range.ipynb – **Cell 2**

Import necessary packages/modules

```
[1]  # Cell 1
     import json
     from pathlib import Path

     import matplotlib.pyplot as plt
     import numpy as np
     from google.colab import drive

     import pandas as pd
```

Connect this notebook to your Google Drive

```
[2]  # Cell 2
     drive.mount("/content/gdrive", force_remount=True)
     notebook_path = Path("/content/gdrive/MyDrive/asp-hs")
     notebook_path = notebook_path / Path("Session 03 - Computing in Science")
     notebook_path
```

```
Mounted at /content/gdrive
PosixPath('/content/gdrive/MyDrive/asp-hs/Session 03 - Computing in Science')
```

Run plot_liquid_range.ipynb – **Cell 3**

Rehydrate/deserialize a Python dictionary from a JSON formatted text file

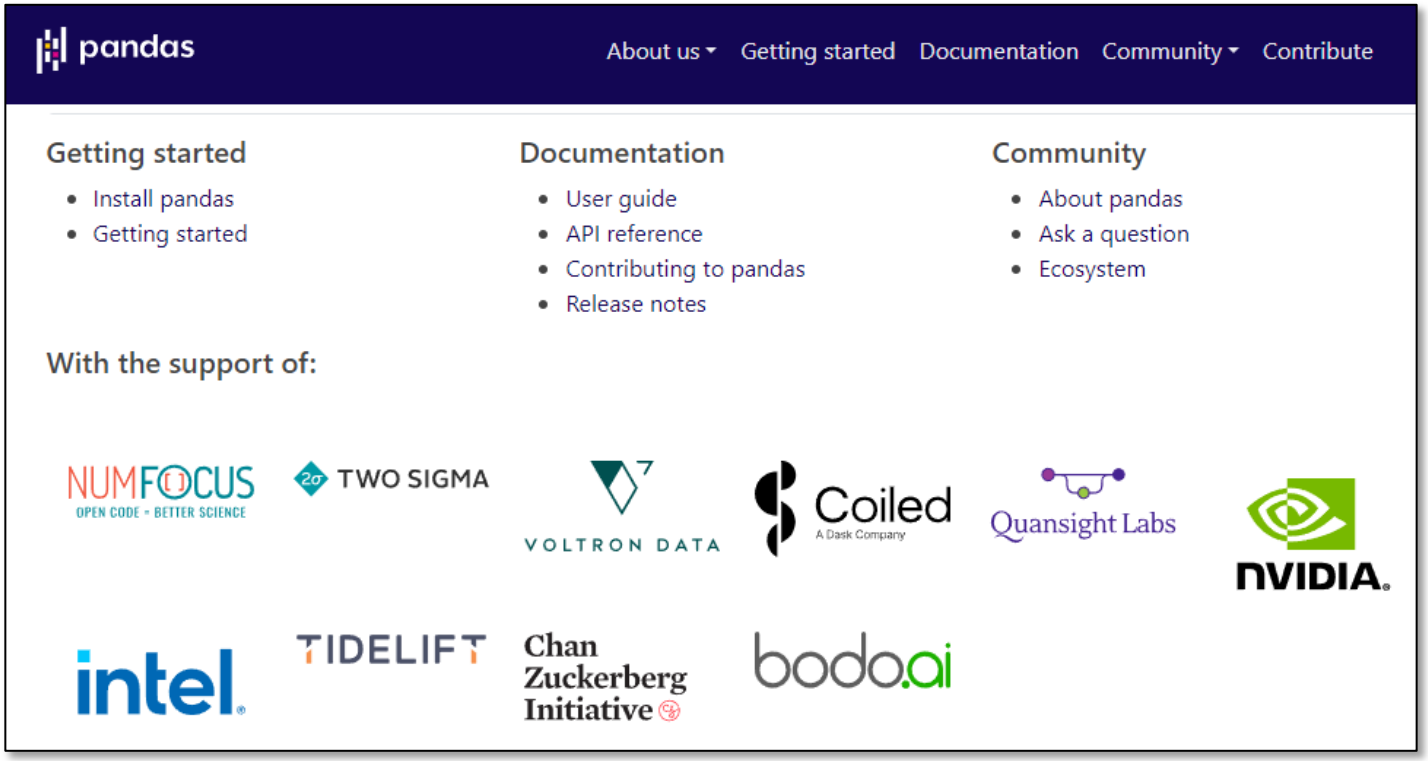
```
[3]  # Cell 3
     file_name = "periodic_table.json"
     file_path = notebook_path/ file_name
     with Path.open(file_path, "rb") as f_in:
         periodic_table = json.load(f_in)
     periodic_table["elements"][0]["name"]
```

```
'Hydrogen'
```

① ② ③ ④

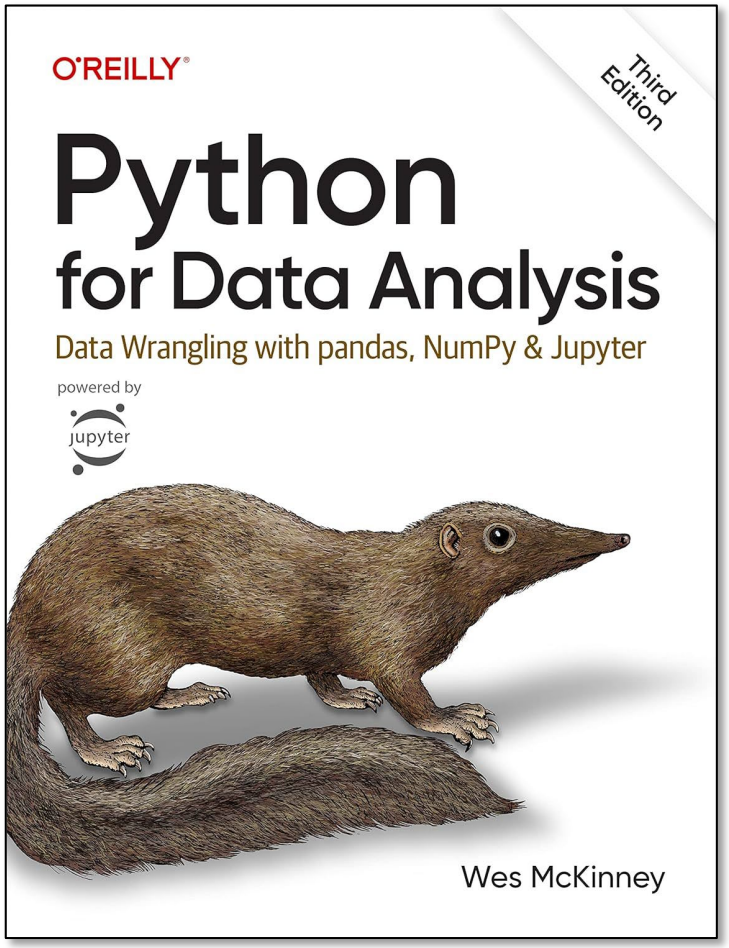
Extending Python via the **pandas** Package

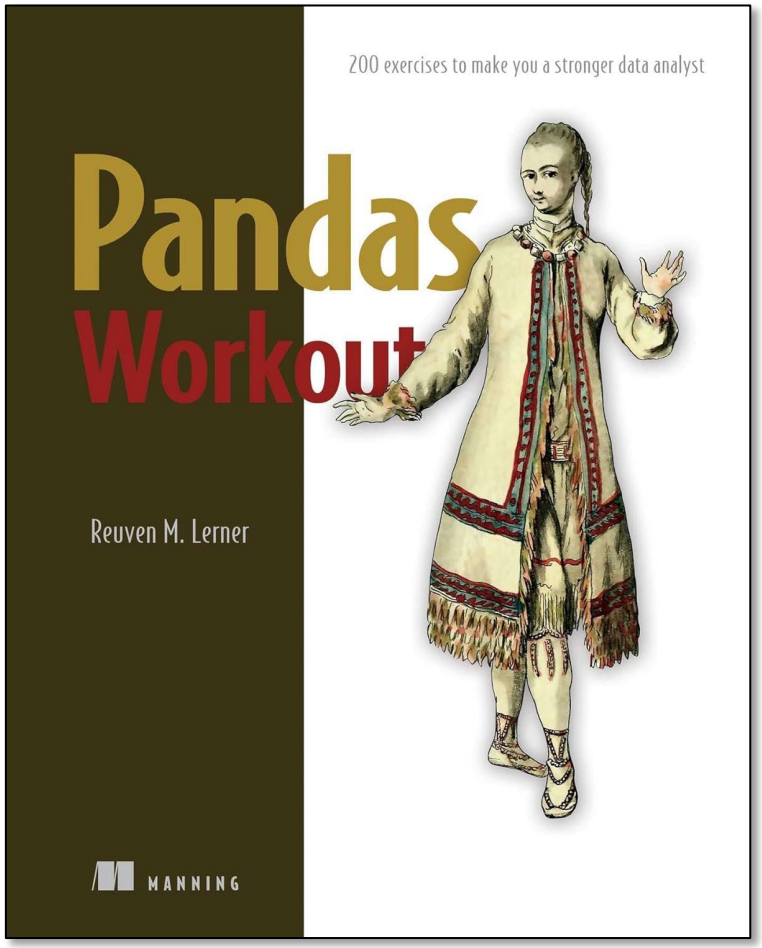
https://pandas.pydata.org



Extending Python via the **pandas** Package

Pandas is 100% Open Source and Free of Cost





Run plot_liquid_range.ipynb – **Cell 4**

Create a Python `list` of elements sorted by group, then period, then atomic number ← ①

1. Use `for` loops to range over every group # and period #
2. Extract the dictionary values for those keys (elements) that match the current group # and period #
3. Append the matching element to the growing `list` of elements
4. Each item in the `elements` list is a tuple containing:
   a. A `string` label formed by of its atomic symbol, a hyphe
   b. The element's melting point (or −∞ if not melting point
   c. The element's boiling point (or ∞ if not melting point is

```
[4]  # Cell 4
     elements: list = []                                    ← ③
     for group in range(1, 19):                             ← ④
         for period in range(1, 8):                         ← ⑤
             for k, v in enumerate(periodic_table["elements
                 if group == int(v["group"]) and period ==
                     elements.append(                       ← ⑦
                         (
                             f"{v['symbol']}-{v['number']}"
                             float(v["melt"] or -np.inf),
                             float(v["boil"] or np.inf),
                         ),
                     )
     pd.DataFrame(elements[:10], columns=["Element","Melt","Boil"])   ← ⑩
```

	Element	Melt	Boil
0	H-1	13.990	20.271
1	Li-3	453.650	1603.000
2	Na-11	370.944	1156.090
3	K-19	336.700	1032.000
4	Rb-37	312.450	961.000
5	Cs-55	301.700	944.000
6	Fr-87	300.000	950.000
7	Be-4	1560.000	2742.000
8	Mg-12	923.000	1363.000
9	Ca-20	1115.000	1757.000

Run plot_liquid_range.ipynb – **Cell 5**

In preperation for plotting the melting and boiling points: ← ①

1. Create a numpy `array` from the elements `list`
2. Create a numpy `array` of the melting points, which are the 2nd item [index #1] in each tuple in the elements list
3. Create a numpy `array` of the boiling points, which are the 3rd item [index #2] in each tuple in the elements list
4. Convert both melting and boiling point values from Kelvin to Celsius ← ②

```
[5]  # Cell 5
     data = np.array(elements)                          ← ③
     melt = np.array(data[:, 1], dtype=float) - 273.15
     boil = np.array(data[:, 2], dtype=float) - 273.15
     pd.DataFrame({                                     ← ⑤
         'Element' : [element[0] for element in elements[
         'Melt': melt[:10],
         'Boil': boil[:10]
     })
```

	Element	Melt	Boil
0	H-1	-259.160	-252.879
1	Li-3	180.500	1329.850
2	Na-11	97.794	882.940
3	K-19	63.550	758.850
4	Rb-37	39.300	687.850
5	Cs-55	28.550	670.850
6	Fr-87	26.850	676.850
7	Be-4	1286.850	2468.850
8	Mg-12	649.850	1089.850
9	Ca-20	841.850	1483.850

Run plot_liquid_range.ipynb – **Cells 6...7**

Find the element with the smallest liquid range (boiling point - melting point)

1. The variable `liquid_range` becomes an numpy `array` because *boil* and *melt* are both arrays
2. The function `np.argmin()` returns the index in the given array with the minimum value

```
[6]  # Cell 6
     liquid_range = boil - melt
     min_idx = np.argmin(liquid_range)
     min_range = liquid_range[min_idx]
     print(f"Smallest liquid range: {min_range:,.2f}°C is {data[min_idx, 0]}")
```

① ←

```
Smallest liquid range: 2.54°C is Ne-10
```

Find the element with the largest liquid range

1. The `liquid_range` array is sorted so the highest values are first
2. The function `np.argwhere().min()` returns the first index where the condition is true
3. Recall that elements without a known boiling point have their boiling point sent to ∞
4. Therefore, we need to find the first index where the maximum liquid range that is NOT ∞

```
[7]  # Cell 7
     measured_liquid_range = np.array(sorted(liquid_range, reverse=True))
     max_measured_idx = np.argwhere(measured_liquid_range < np.inf).min()
     max_range = measured_liquid_range[max_measured_idx]
     max_idx = np.argwhere(liquid_range == max_range)[0, 0]
     print(f" Largest liquid range: {max_range:,.2f}°C is {data[max_idx, 0]}")
```

② ←

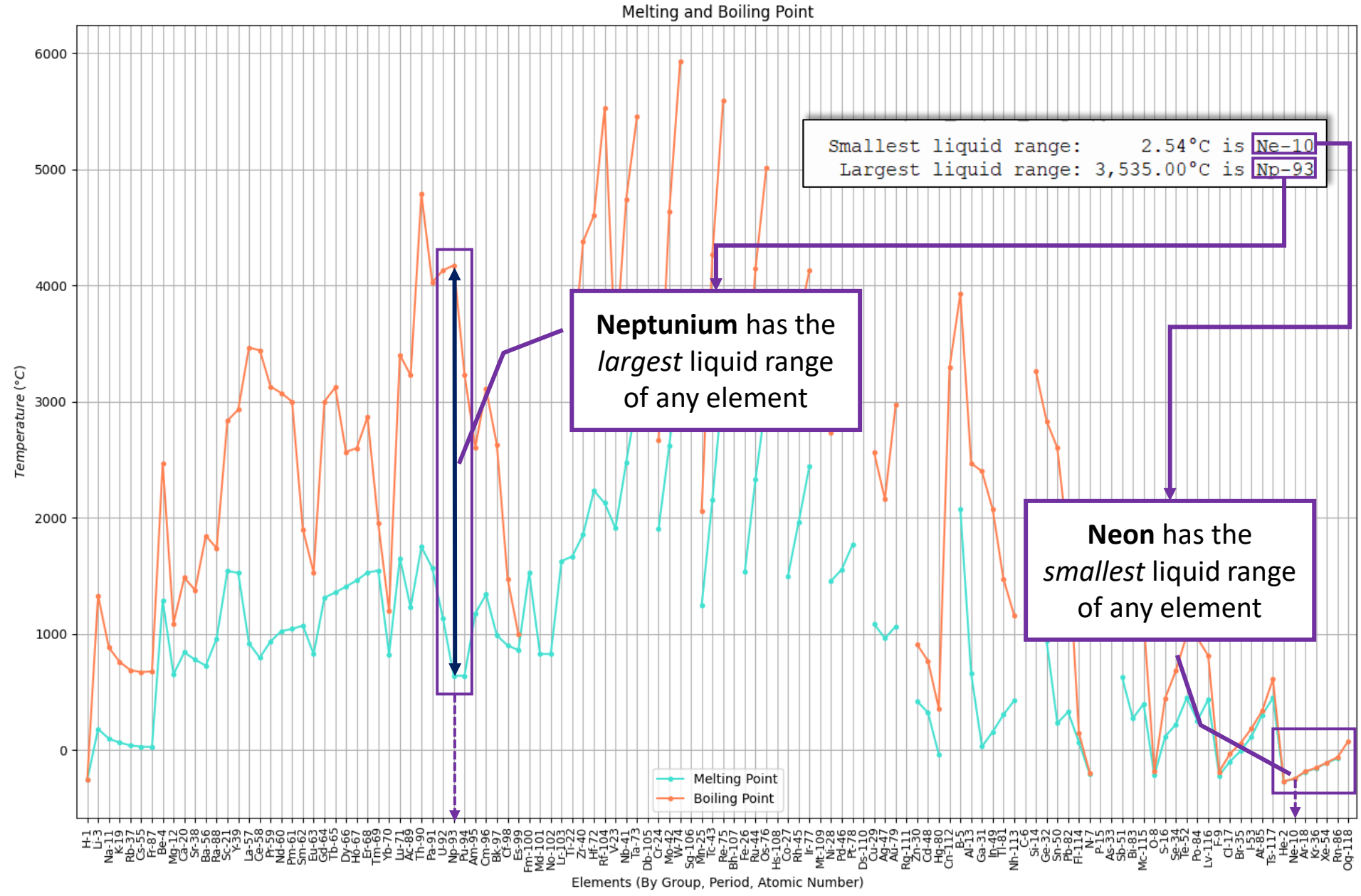
```
 Largest liquid range: 3,535.00°C is Np-93
```

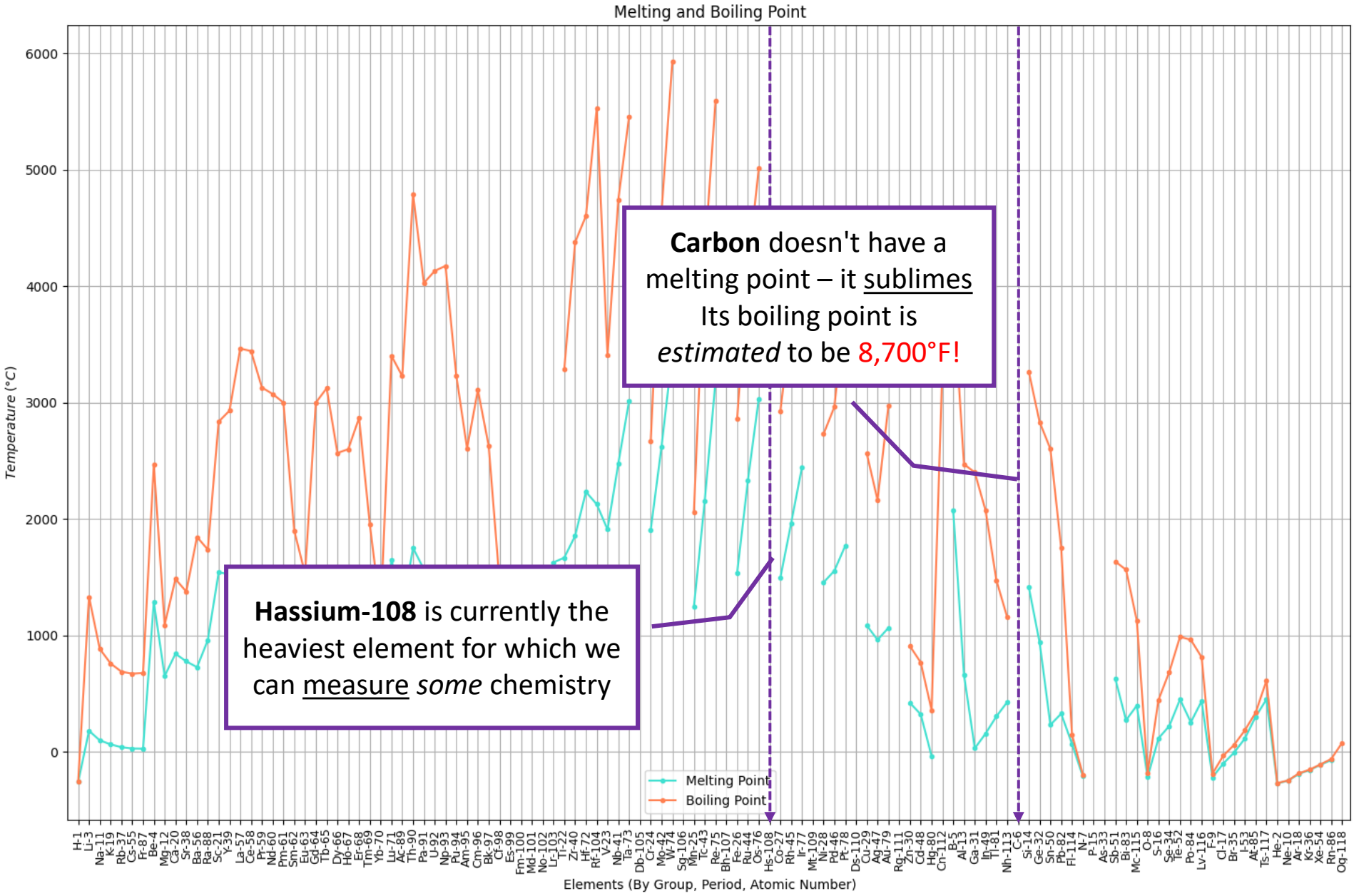
Run plot_liquid_range.ipynb – **Cell 8**

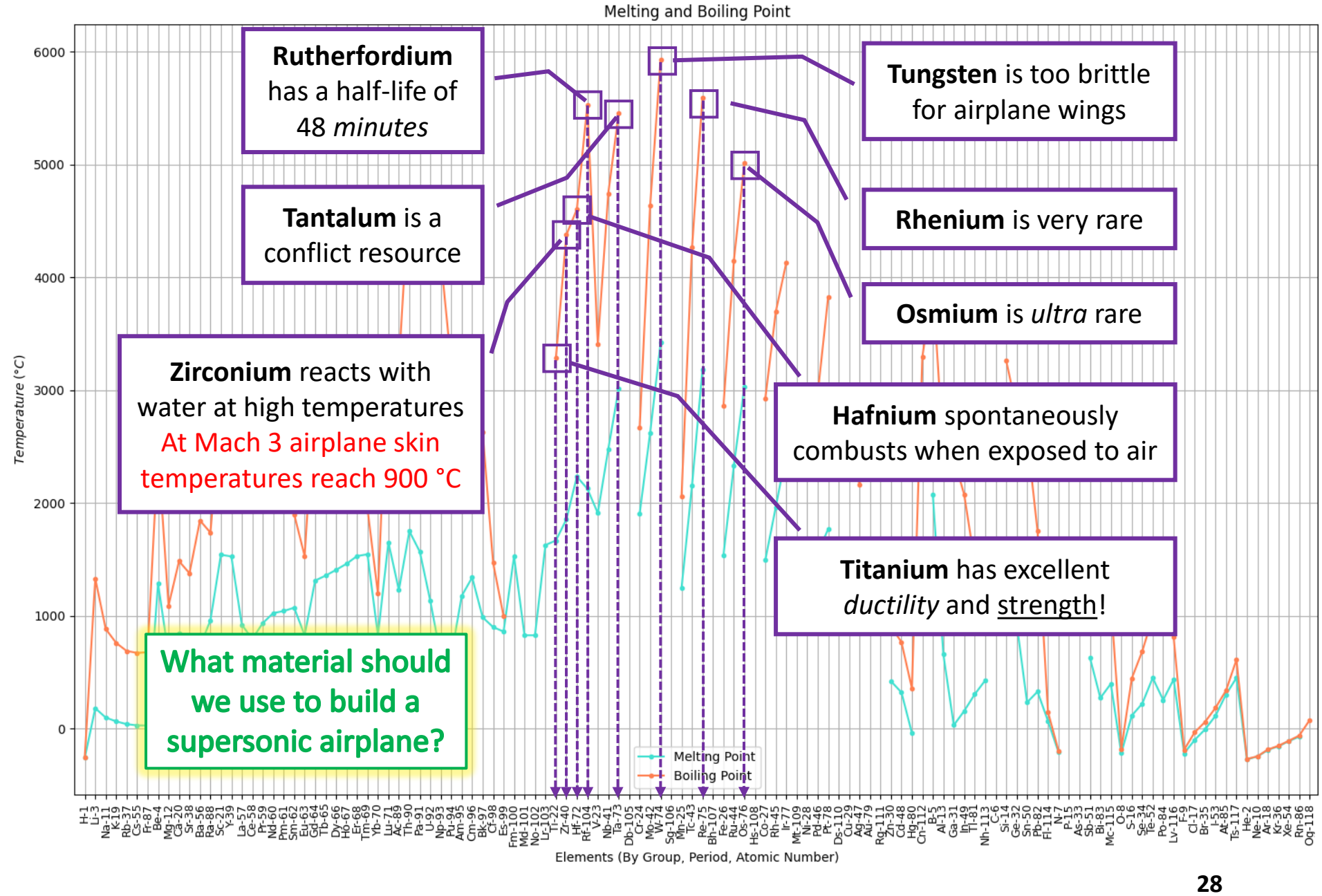
Plot the melting point and boiling point (if available) for every element sorted by group, period, atomic number

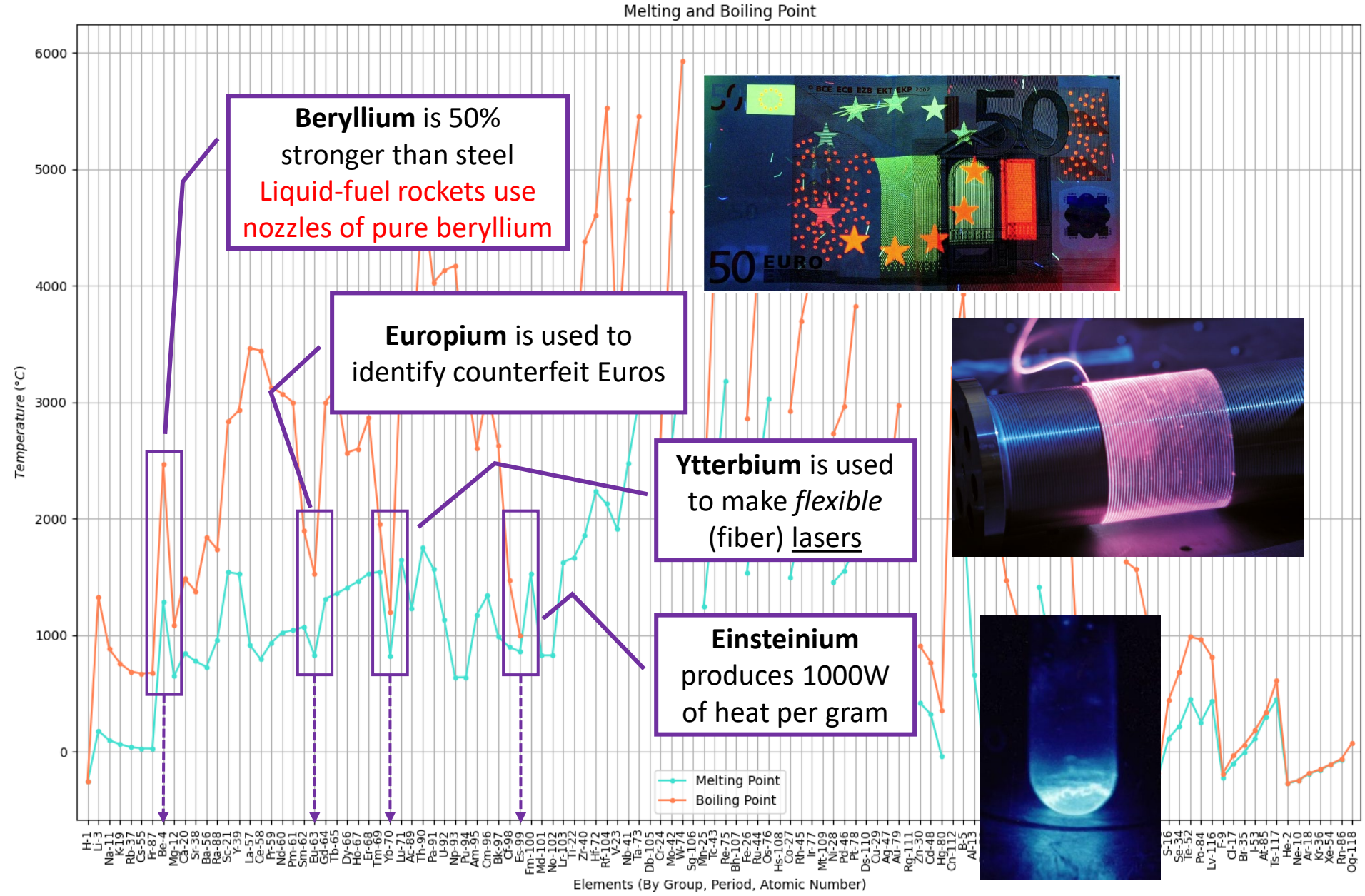
```
[8]  # Cell 8
     plt.figure(figsize=(20, 8))
     x = np.arange(len(elements))
     plt.plot(x, melt, color="turquoise", marker=".", label="Melting Point")
     plt.plot(x, boil, color="coral", marker=".", label="Boiling Point")
     plt.title("Melting and Boiling Point")
     plt.xlabel("Elements (By Group, Period, Atomic Number)")
     plt.ylabel("Temperature (C)")
     ax = plt.gca()
     ax.set_xticks(x)
     ax.set_xticklabels(data[:, 0], fontsize=9, rotation=90)
     ax.legend(loc="lower center")
     ax.grid("on")
     plt.show()
```

① ②


Melting and Boiling Point
Temperature (°C)
Elements (By Group, Period, Atomic Number)
Melting Point
Boiling Point
Smallest liquid range: 2.54°C is Ne-10
Largest liquid range: 3,535.00°C is Np-93
Neptunium has the largest liquid range of any element
Neon has the smallest liquid range of any element

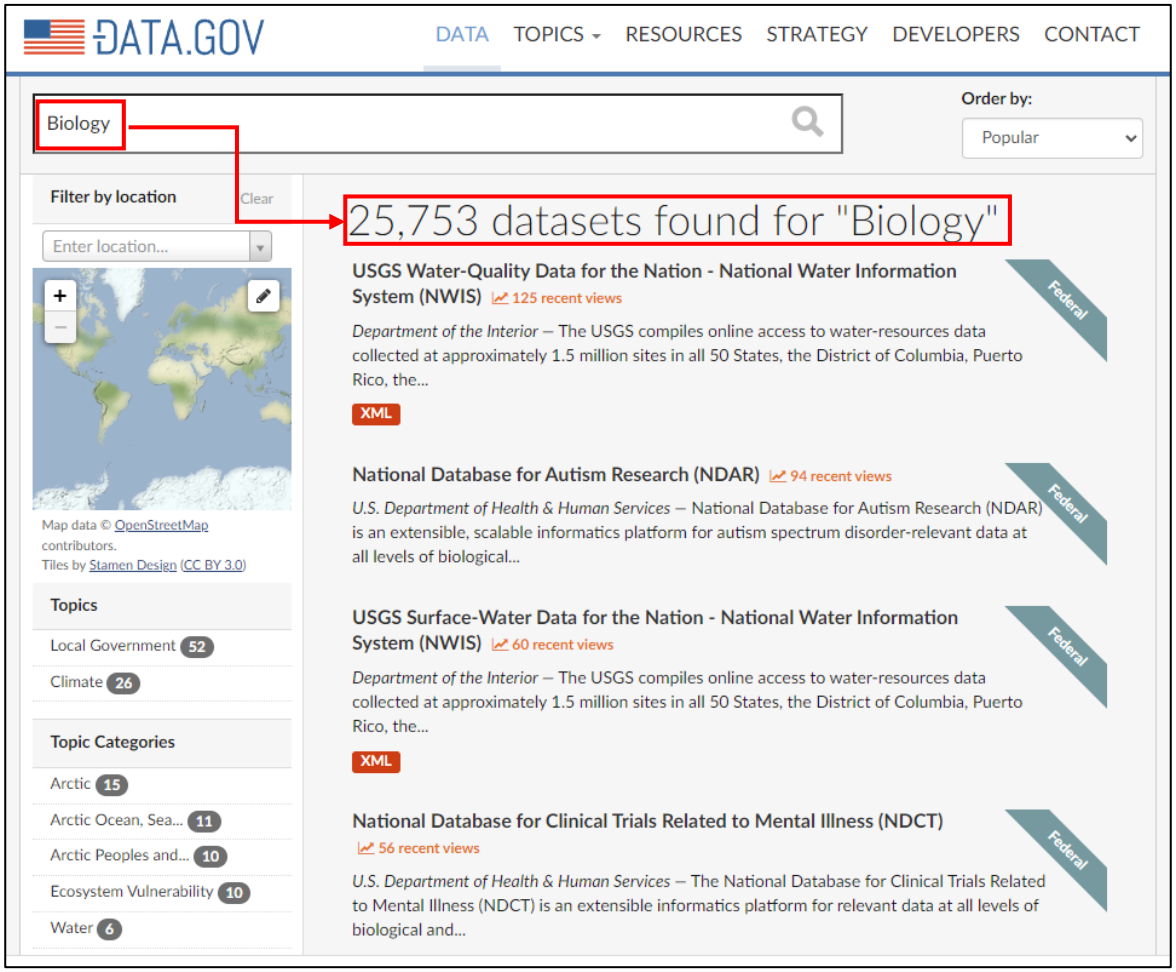

Melting and Boiling Point
Temperature (°C)
6000
5000
4000
3000
2000
1000
0
Carbon doesn't have a melting point – it sublimes
Its boiling point is *estimated* to be 8,700°F!
Hassium-108 is currently the heaviest element for which we can measure *some* chemistry
Melting Point
Boiling Point
Elements (By Group, Period, Atomic Number)


Melting and Boiling Point
Rutherfordium has a half-life of 48 minutes
Tungsten is too brittle for airplane wings
Tantalum is a conflict resource
Rhenium is very rare
Osmium is ultra rare
Zirconium reacts with water at high temperatures
At Mach 3 airplane skin temperatures reach 900 °C
Hafnium spontaneously combusts when exposed to air
Titanium has excellent ductility and strength!
What material should we use to build a supersonic airplane?
Melting Point
Boiling Point
Temperature (°C)
Elements (By Group, Period, Atomic Number)


Melting and Boiling Point
Temperature (°C)
6000
5000
4000
3000
2000
1000
0
Melting Point
Boiling Point
Elements (By Group, Period, Atomic Number)
Beryllium is 50% stronger than steel
Liquid-fuel rockets use nozzles of pure beryllium
Europium is used to identify counterfeit Euros
Ytterbium is used to make flexible (fiber) lasers
Einsteinium produces 1000W of heat per gram
50 EURO

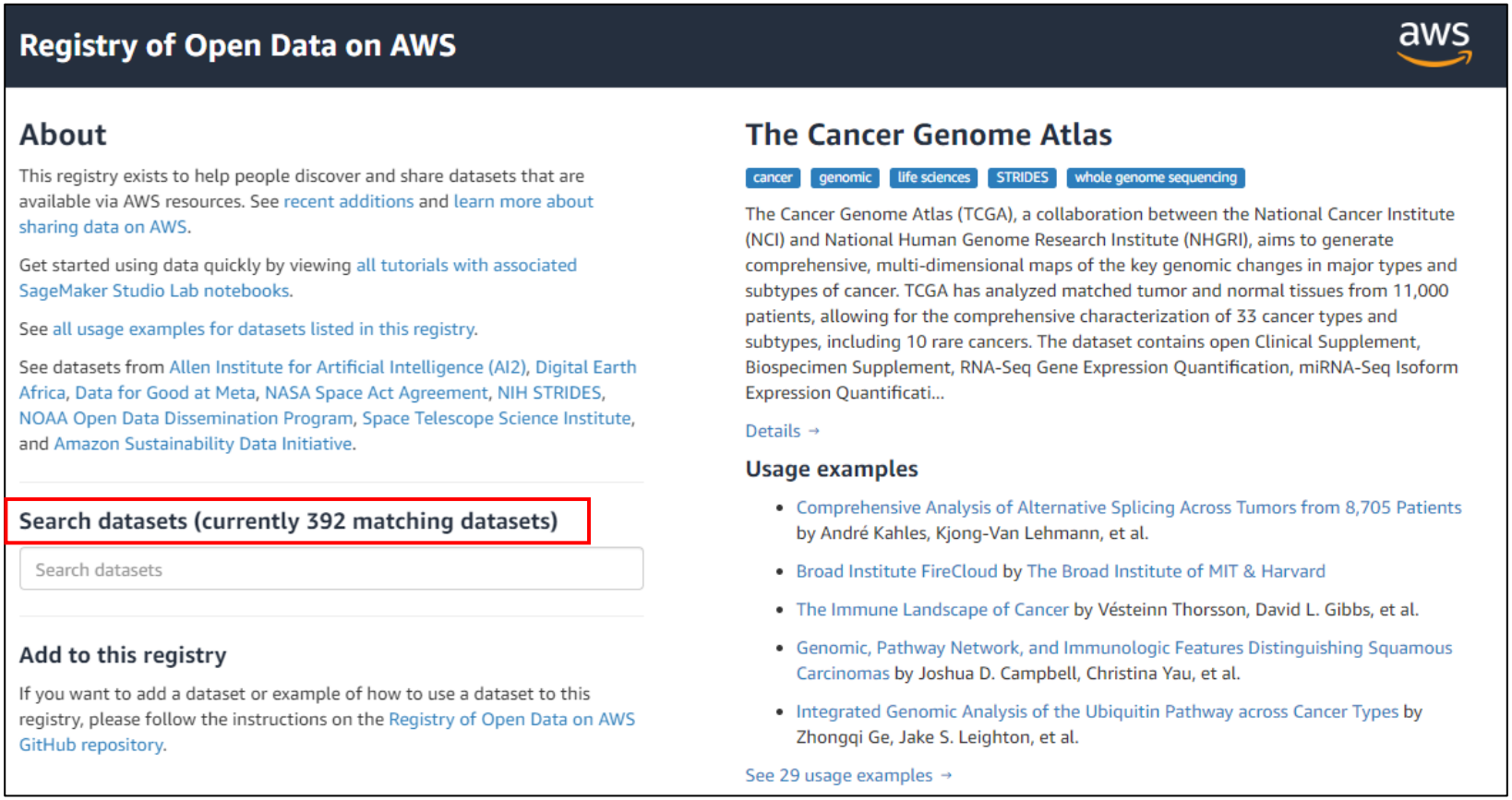
DATA.GOV

https://catalog.data.gov



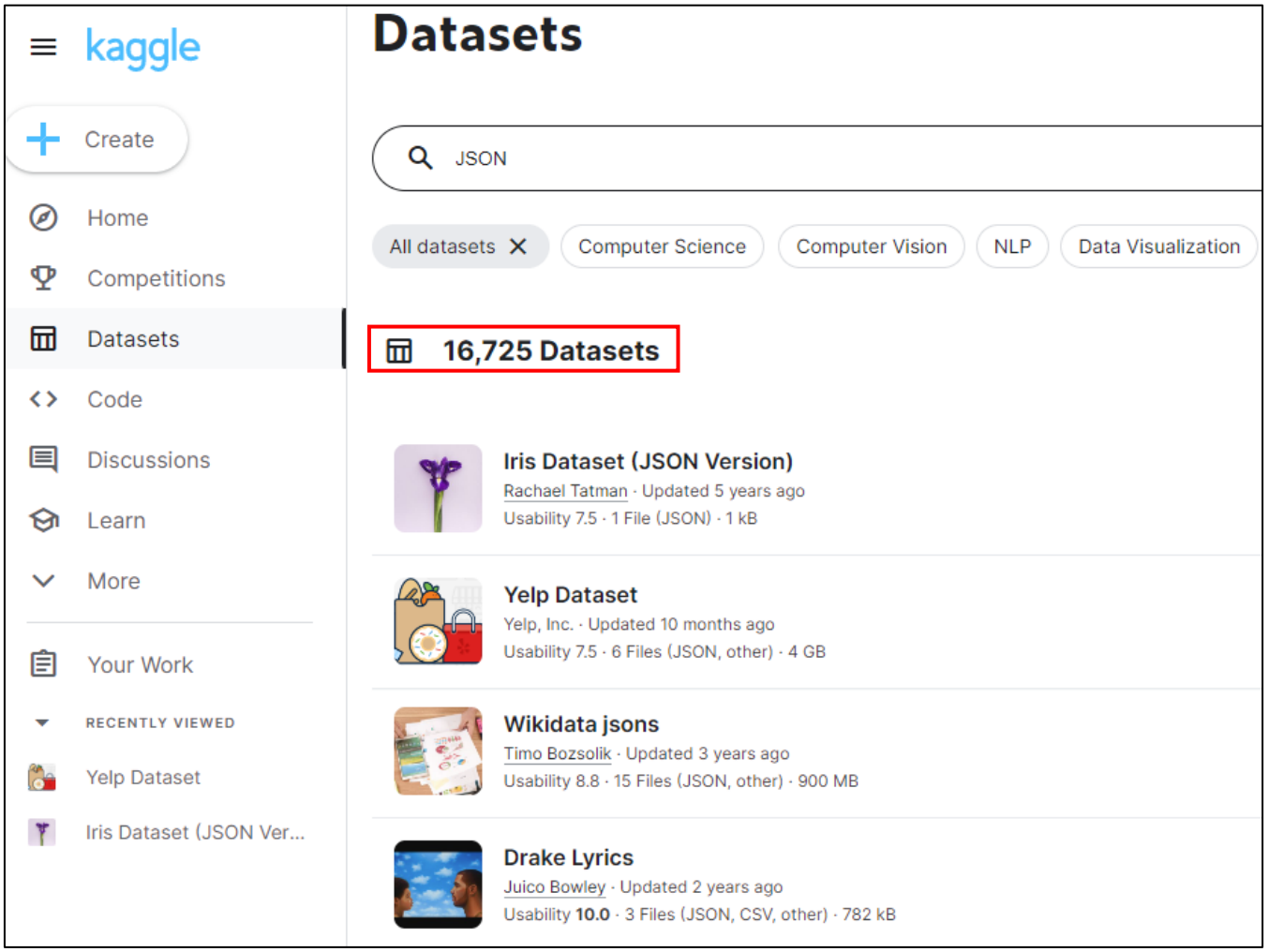
Amazon Open Data Sets

https://registry.opendata.aws



Kaggle

https://www.kaggle.com/datasets



Identify an Unknown Element

Your lab assistant distilled 50 g of a gas into a cylinder, but he left without writing down what kind of gas it is. The cylinder has a pressure regulator that adjusts a piston to keep the pressure at a constant 2.00 atm. To identify the gas, you measure the cylinder volume at several different temperatures, acquiring the data shown at the right. What is the gas?

T (°C)	V (L)
−50	11.6
0	14.0
50	16.2
100	19.4
150	21.8

$$PV = nRT$$

SI Units

constant — P = pressure — pascals (Pa)

V = volume — meters^3

constant — n = amount of substance

constant — R = ideal gas constant

T = temperature — kelvin (K)

$$V = \left(\frac{nR}{P}\right)T$$

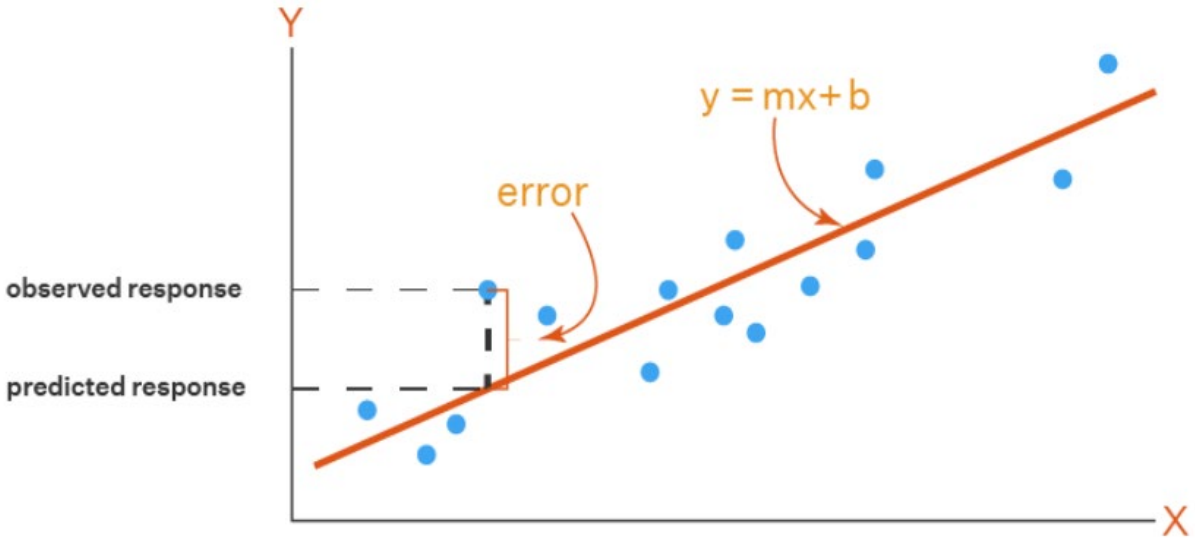
$y = mx$

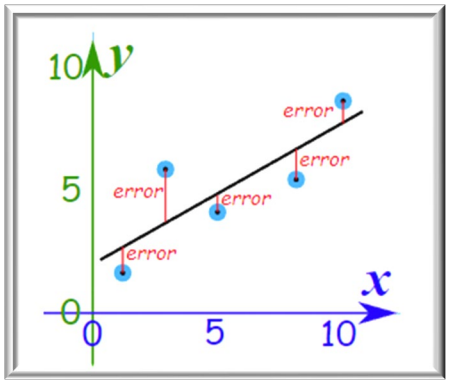
$$V = (m)T$$

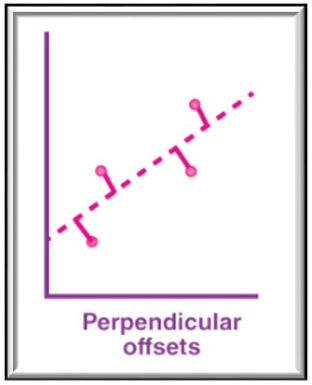
constant

Method of Least Squares

Carl Friedrich
Gauss
(1777-1855)







Run identify_element.ipynb – **Cells 1...2**

Import necessary packages/modules

```
[1]  # Cell 1
     from pathlib import Path

     import matplotlib.pyplot as plt
     import numpy as np
     from google.colab import drive

     import pandas as pd
```

① ←

Connect this notebook to your Google Drive

```
[2]  # Cell 2
     drive.mount("/content/gdrive", force_remount=True)
     notebook_path = Path("/content/gdrive/MyDrive/asp-hs")
     notebook_path = notebook_path / Path("Session 03 - Computing in Science")
     notebook_path
```

② ←

```
Mounted at /content/gdrive
PosixPath('/content/gdrive/MyDrive/asp-hs/Session 03 - Computing in Science')
```

Run identify_element.ipynb – **Cell 3**

Generate a `numpy array` from a CSV (comma separated value) formatted text file

1. The 1st column (index 0) is the temperature in ° Celsius
2. The 2nd column (index 1) is the volume in Liters

```
[3]  # Cell 3
     file_name = "gas.csv"                                          ①
     file_path = notebook_path/ file_name
     data = np.genfromtxt(file_path, delimiter=",")                 ②
     pd.DataFrame(data, columns=["Temperature (C)","Volume (L)"])   ③
```

	Temperature (C)	Volume (L)
0	-50.0	11.6
1	0.0	14.0
2	50.0	16.2
3	100.0	19.4
4	150.0	21.8

④

Run identify_element.ipynb – **Cell 4**

Convert the experiment data to SI units

1. Temperature should be in ° Kelvin
2. Volume should be in cubic Meters

① ←

```
[4]  # Cell 4
     temperature = data[:, 0] + 273.15  # 1st column to kelvin
     volume = data[:, 1] / 1000  # 2nd column to meters cubed
     pd.DataFrame({
         "Temperature (K)": temperature,
         "Volume (m^3)": volume
     })
```

② ←

③ ←

④ ←

	Temperature (K)	Volume (m^3)
0	223.15	0.0116
1	273.15	0.0140
2	323.15	0.0162
3	373.15	0.0194
4	423.15	0.0218

⑤ ←

Run identify_element.ipynb – **Cell 5**

Define a function to calculate the line of best fit through the points (x, y)

1. Use Gauss's Linear Regression formulas that minimize the error ← ①
2. The (x) array holds the independent variable values
3. The (y) array holds the dependent variable values
4. The function returns m (slope) and b (y-intercept) of the line of best fit passing through the (x, y) points

```
[5]  # Cell 5
     def fit_linear(x, y):                                   ← ②
         m = len(x) * np.sum(x * y) - np.sum(x) * np.sum(y)
         m = m / (len(x) * np.sum(x**2) - np.sum(x) ** 2)
         b = (np.sum(y) - m * np.sum(x)) / len(x)
         return m, b                                         ← ③

     # Calculate line of best fit
     slope, yint = fit_linear(temperature, volume)           ← ④
     print(f"slope: {slope:.8f}")
     print(f"y-intercept: {yint:.8f}")
```

```
slope: 0.00005160                                            ← ⑤
y-intercept: -0.00007454
```

$$m = \frac{n\sum(XY) - \sum Y \sum X}{n\sum(X^2) - (\sum X)^2}$$

$$b = \frac{\sum Y - m\sum X}{n}$$

Slope of line of best fit $m = \frac{\Delta y}{\Delta x} = \frac{V}{T}$

Run identify_element.ipynb – **Cell 6…7**

Leverage the ideal gas law $PV = nRT$ to solve for n (number of moles of the unknown gas)

```
[6]  # Cell 6
     p = 2.0 * 101_325  # Convert 2.0 atm (given) to pascals
     r = 8.31446261815324  # Gas constant (SI units)
     n = p / r * slope  # Moles of gas (rearrange ideal gas law equation)
     print(f"Number of moles of gas: {n:.8f}")
```

① ②

```
Number of moles of gas: 1.25765675
```

$$PV = nRT \quad n = \left(\frac{P}{R}\right)\left(\frac{V}{T}\right) = \frac{P}{R}(m)$$

Calculate the atomic mass of the homogenous (pure) gas in the cylinder

```
[7]  # Cell 7
     m_sample = 50  # (given) grams
     atomic_mass = m_sample / n  # sample mass divided by number of moles
     print(f"Atomic mass of unknown gas: {atomic_mass:8f}u")
```

③

```
Atomic mass of unknown gas: 39.756476u
```

④

Molar Mass = Atomic Mass (specific isotope) ≈ Atomic Weight (avg)

Run identify_element.ipynb – **Cell 8**

In preparation for plotting the Temperature vs. Volume line for this gas:

1. Create a linear space of temperature (t) values spanning 0 to 500° Kelvin ← ①
2. Display the numpy `array` (t)

```
[8]  # Cell 8
     t = np.linspace(0, 500)   ← ②
     display(t)
```

```
array([  0.        ,  10.20408163,  20.40816327,  30.6122449 ,
        40.81632653,  51.02040816,  61.2244898 ,  71.42857143,
        81.63265306,  91.83673469, 102.04081633, 112.24489796,
       122.44897959, 132.65306122, 142.85714286, 153.06122449,
       163.26530612, 173.46938776, 183.67346939, 193.87755102,
       204.08163265, 214.28571429, 224.48979592, 234.69387755,
       244.89795918, 255.10204082, 265.30612245, 275.51020408,
       285.71428571, 295.91836735, 306.12244898, 316.32653061,
       326.53061224, 336.73469388, 346.93877551, 357.14285714,
       367.34693878, 377.55102041, 387.75510204, 397.95918367,
       408.16326531, 418.36734694, 428.57142857, 438.7755102 ,
       448.97959184, 459.18367347, 469.3877551 , 479.59183673,
       489.79591837, 500.        ])
```

← ③

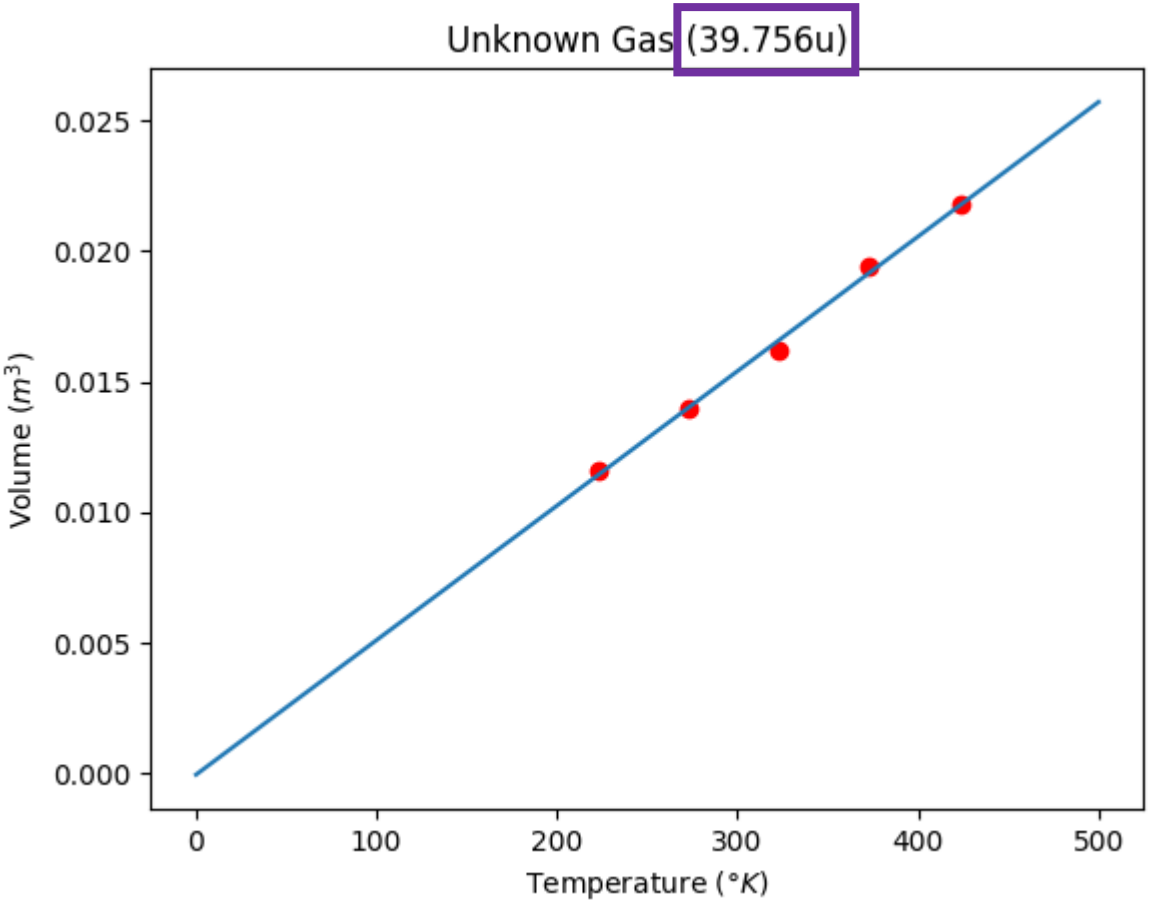
Run identify_element.ipynb – **Cell 9**

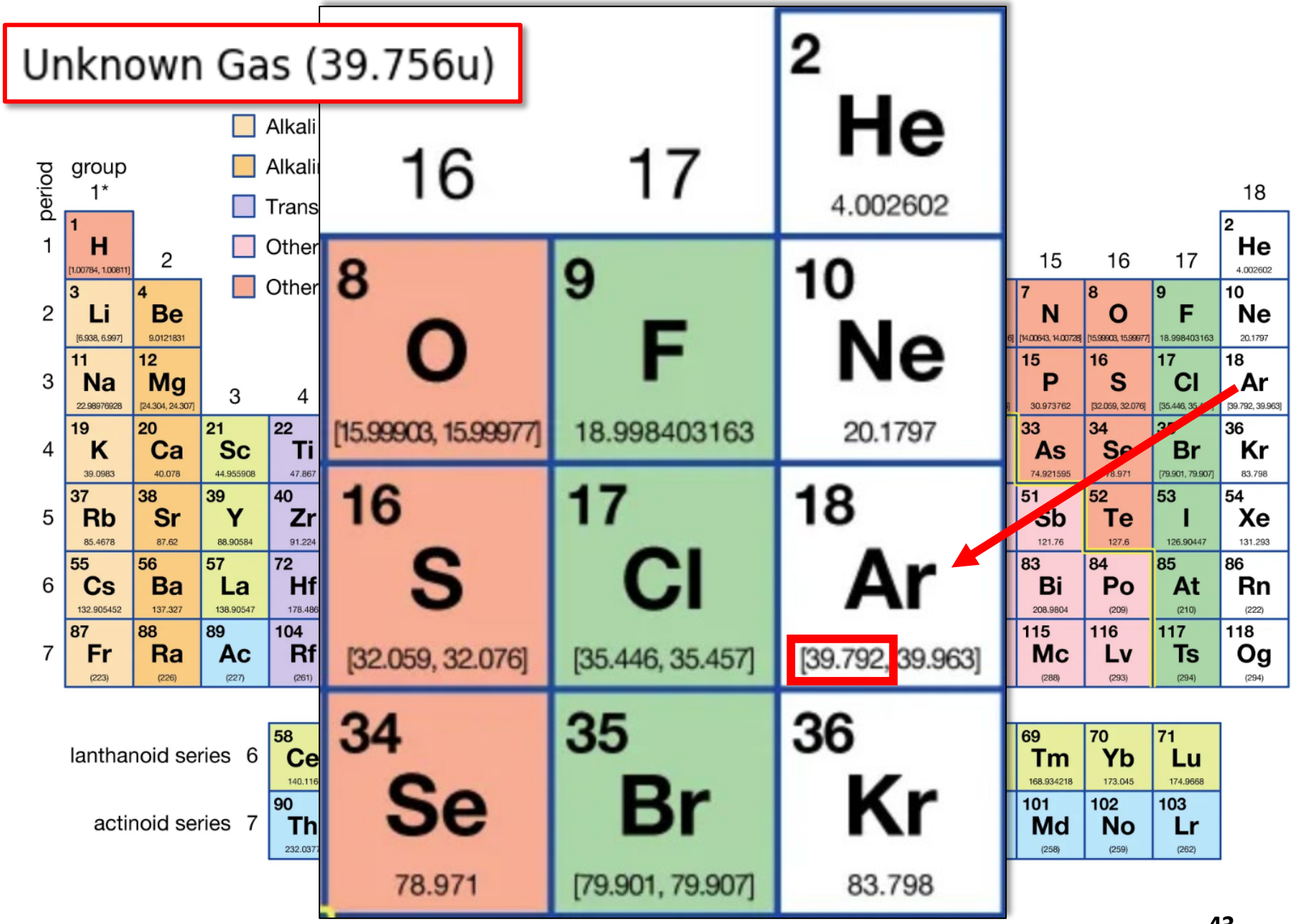
Plot the Temperature vs. Volume line for this gas ← ①

1. Superimpose the data points from the CSV file
2. Display the atomic mass of the unknown element in the graph's title

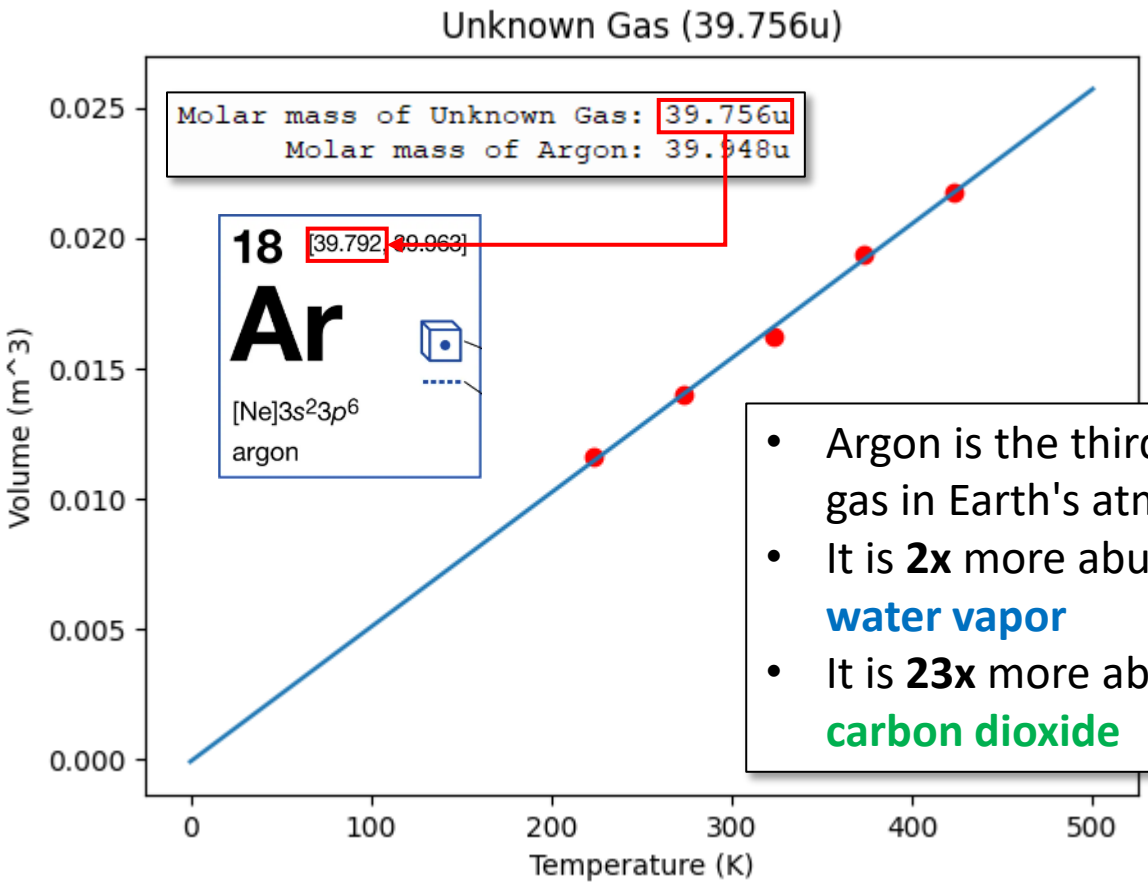
```
[9]  # Cell 9
     plt.scatter(temperature, volume, color="red")   ← ②
     plt.plot(t, slope * t + yint)                   ← ③
     plt.title(f"Unknown Gas ({atomic_mass:.3f}u)")
     plt.xlabel(r"Temperature $(\degree K)$")
     plt.ylabel("Volume ($m^3$)")
     plt.show()
```

View identify_element.ipynb – **Cell 9**




Unknown Gas (39.756u)
16
17
2
He
4.002602
8
O
[15.99903, 15.99977]
9
F
18.998403163
10
Ne
20.1797
16
S
[32.059, 32.076]
17
Cl
[35.446, 35.457]
18
Ar
[39.792, 39.963]
34
Se
78.971
35
Br
[79.901, 79.907]
36
Kr
83.798
Alkali
Alkali
Trans
Other
Other
group
period
lanthanoid series
actinoid series

Check identify_element.ipynb – **Cell 9**

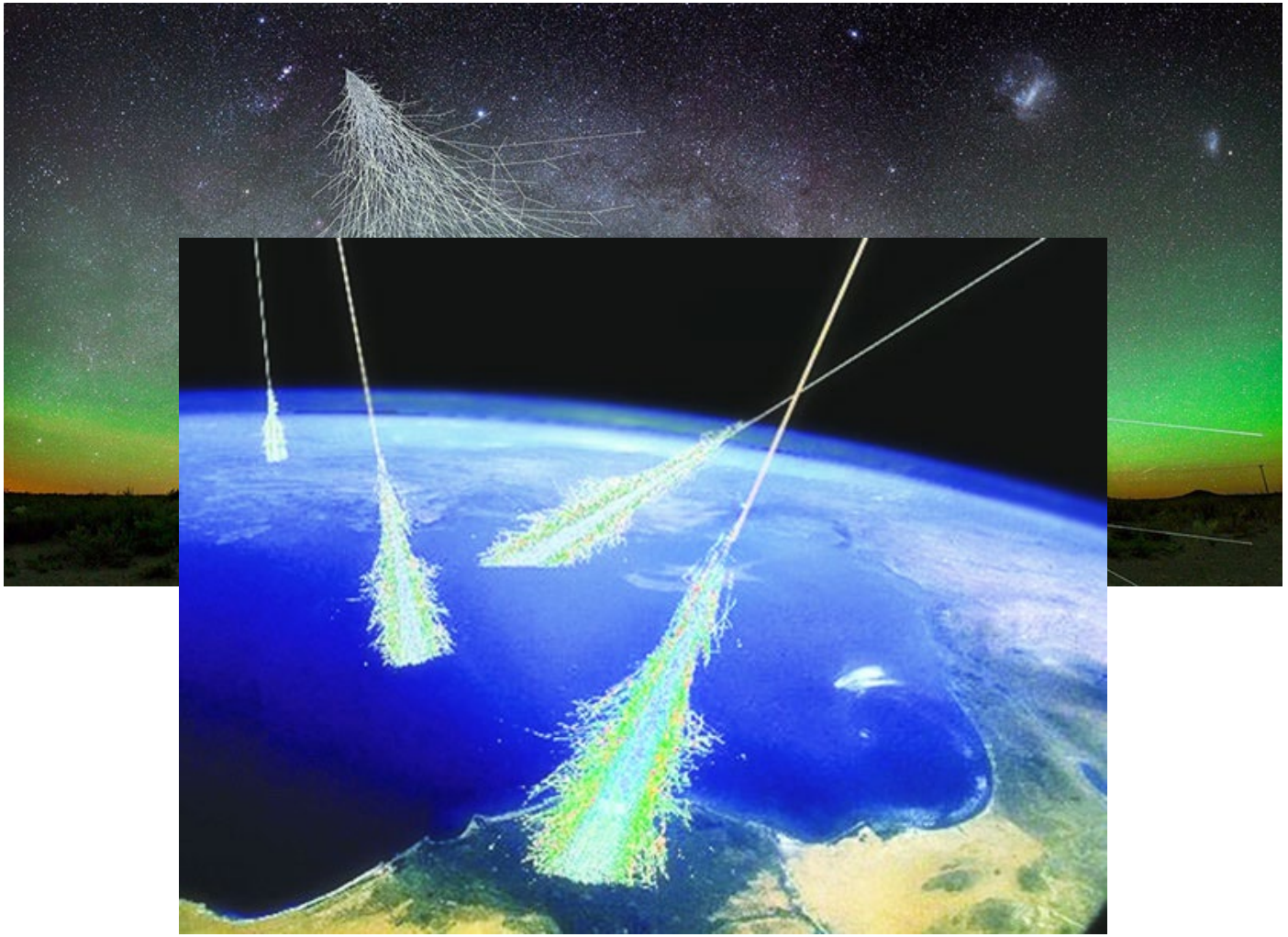


- Argon is the third-most abundant gas in Earth's atmosphere
- It is **2x** more abundant than **water vapor**
- It is **23x** more abundant than **carbon dioxide**

Cosmic Rays

- Cosmic rays entering the Earth's atmosphere collide with gas molecules, creating secondary particles
  - Your scientist has developed an instrument to capture the trajectory of these secondary particles as they rain down

Cosmic Rays



Cosmic Rays

- Cosmic rays entering the Earth's atmosphere collide with gas molecules, creating secondary particles
  - Your scientist has developed an instrument to capture the trajectory of these secondary particles as they rain down
  - He has given you a data file (**ray.csv**) of a particle's height (in **centimeters**) over the final **nanoseconds** before its impact

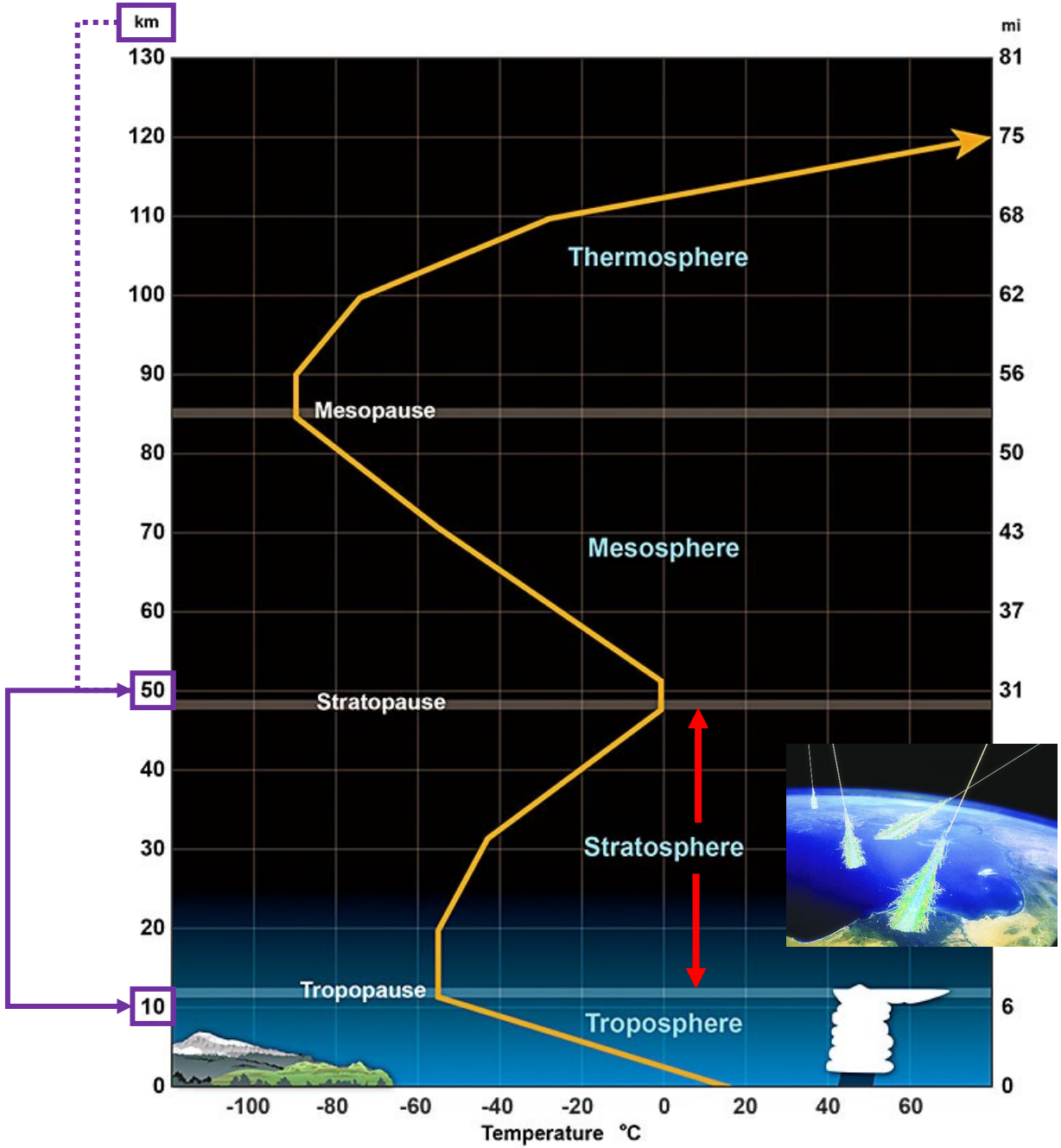
Cosmic Rays

ray.csv

	A	B
1	# ns	cm
2	0	2370
3	1	2345.9335
4	2	2323.7895
5	3	2299.9575
6	4	2274.3289
7	5	2249.8281
8	6	2228.9797
9	7	2199.7143
10	8	2183.1641
11	9	2164.6525
12	10	2127.9649
13	11	2119.1754
14	12	2089.6137
15	13	2051.3933
16	14	2036.5715
17	15	2026.0944
18	16	1997.1185
19	17	1961.2001
20	18	1951.8214

Cosmic Rays

- Cosmic rays entering the Earth's atmosphere collide with gas molecules, creating secondary particles
  - Your scientist has developed an instrument to capture the trajectory of these secondary particles as they rain down
  - He has given you a data file (**ray.csv**) of a particle's height (in **centimeters**) over the final **nanoseconds** before its impact
  - The scientist knows the secondary particle was **not** accelerating and lived for only **0.1743 milliseconds**
  - He wants you to determine its velocity (relative to ***c***) and the height (in **km**) in the *stratosphere* at which it was originally emitted
- How would you display the particle's path and **use the line of best fit** to determine those *two* unknowns?


km
mi
130
120
110
100
90
80
70
60
50
40
30
20
10
0
81
75
68
62
56
50
43
37
31
6
0
Thermosphere
Mesopause
Mesosphere
Stratopause
Stratosphere
Tropopause
Troposphere
-100
-80
-60
-40
-20
0
20
40
60
Temperature °C

Run cosmic_rays.ipynb – **Cells 1...2**

Import necessary packages/modules

```
[1]  # Cell 1
     from pathlib import Path

     import matplotlib.pyplot as plt
     import numpy as np
     from google.colab import drive

     import pandas as pd
```

① ←

Connect this notebook to your Google Drive

```
[2]  # Cell 2
     drive.mount("/content/gdrive", force_remount=True)
     notebook_path = Path("/content/gdrive/MyDrive/asp-hs")
     notebook_path = notebook_path / Path("Session 03 - Computing in Science")
     notebook_path
```

② ←

```
Mounted at /content/gdrive
PosixPath('/content/gdrive/MyDrive/asp-hs/Session 03 - Computing in Science')
```

Run cosmic_rays.ipynb – **Cell 3**

Generate a `numpy array` from a CSV (comma separated value) formatted text file ①

Print the first 10 rows in the generated array

```
[3]  # Cell 3
     file_name = "ray.csv"                                          ②
     file_path = notebook_path/ file_name
     data = np.genfromtxt(file_path, delimiter=",")                 ③
     pd.DataFrame(data[:10], columns=["Time (ns)","Height (cm)"])   ④
```

	Time (ns)	Height (cm)
0	0.0	2370.0000
1	1.0	2345.9335
2	2.0	2323.7895
3	3.0	2299.9575
4	4.0	2274.3289
5	5.0	2249.8281
6	6.0	2228.9797
7	7.0	2199.7143
8	8.0	2183.1641
9	9.0	2164.6525

⑤

Run cosmic_rays.ipynb – **Cell 4**

Slice the data array by columns into two 1D arrays ← ①

1. The 1st column (index 0) is the time (t) in seconds - the *independent* variable ← ②
2. The 2nd column (index 1) is the height (h) above ground level in centimeters - the *dependent* variable
3. Print the first five elements in each array ← ③

```
[4]  # Cell 4
     t = data[:, 0]
     h = data[:, 1]
     print(f"{t[:5] = }")
     print(f"{h[:5] = }")
```

← ④

← ⑤

```
     t[:5] = array([0., 1., 2., 3., 4.])
     h[:5] = array([2370.    , 2345.9335, 2323.7895, 2299.9575, 2274.3289])
```

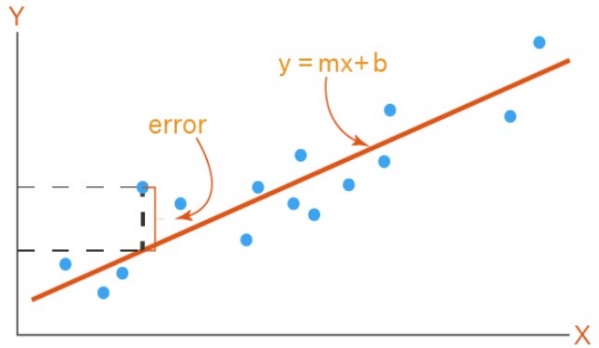
← ⑥

Run cosmic_rays.ipynb – **Cell 5**

Define a function to calculate the line of best fit through the points (x, y)

1. Use Gauss's Linear Regression formulas that minimize the error ← ①
2. The (x) array holds the independent variable values
3. The (y) array holds the dependent variable values
4. The function returns m (slope) and b (y-intercept) of the line of best fit passing through the (x, y) points

```
[5]  # Cell 5
     def fit_linear(x, y):                    ← ②
         m = len(x) * np.sum(x * y) - np.sum(x) * np.sum(y)
         m = m / (len(x) * np.sum(x**2) - np.sum(x) ** 2)
         b = (np.sum(y) - m * np.sum(x)) / len(x)
         return m, b

     # Calculate line of best fit
     slope, yint = fit_linear(t, h)           ← ③
     print(f"slope: {slope:.8f}")
     print(f"y-intercept: {yint:.8f}")
```




```
slope: -23.80824990
y-intercept: 2369.97885028
```
← ④

Run cosmic_rays.ipynb – **Cell 6**

Calculate the particle's origination height oh and initial velocity v ①

1. The origination height should be in kilometers (km) ②
2. The velocity should be relative to the speed of light (c)
3. The radioactive particle existed for only $0.1743\ ms$ before it decayed ③
4. The speed of light is $29.98\ cm/ns$ ④

```
[6]  # Cell 6
     oh = (slope * 1e9 / 100) * (0.1743 / 1e3) / 1000
     v = slope / 29.98                                    ⑤
     print(f"Velocity = {v:.2f}c")
     print(f"Origination Height = {oh:,.2f} km")
```

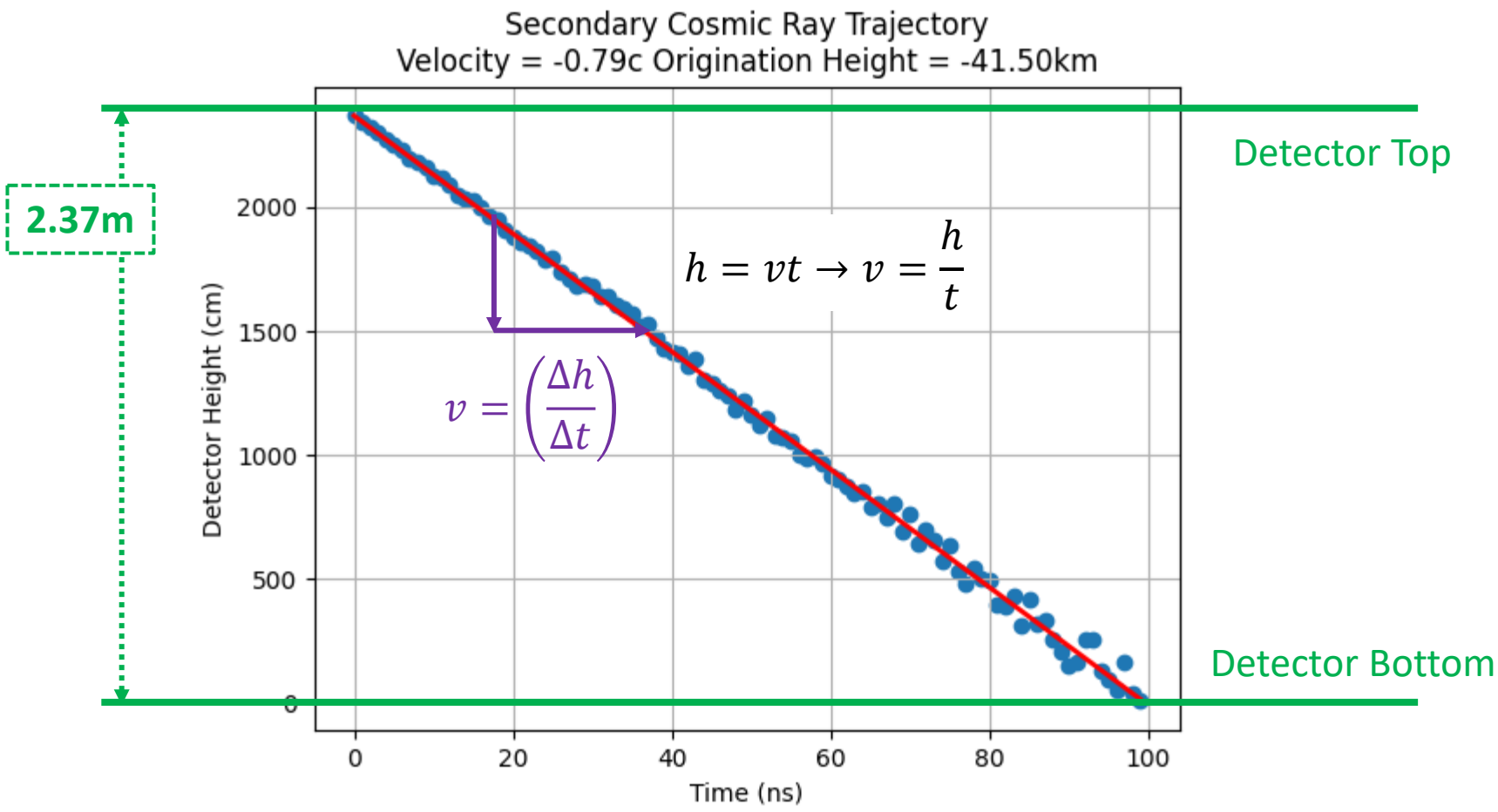
```
Velocity = -0.79c                                         ⑥
Origination Height = -41.50 km
```

Run cosmic_rays.ipynb – **Cell 7**

Plot the trajectory of the particle in the detector as height (cm) vs. time (ns) ← ①

```
[7]  # Cell 7
     plt.figure(figsize=(12, 8))
     plt.scatter(t, h)                                          ← ②
     plt.plot(t, slope * t + yint, color="red", linewidth=2)    ← ③
     plt.title(
         "Secondary Cosmic Ray Trajectory\n"
         f"Velocity = {v:.2f}c "                                ← ④
         f"Origination Height = {oh:,.2f}km",
     )
     plt.xlabel("Time (ns)")
     plt.ylabel("Detector Height (cm)")                         ← ⑤
     plt.grid("on")
     plt.show()
```

View cosmic_rays.ipynb – **Cell 7**



Dimensional Analysis

```
# Calculate origination height (oh) and initial velocity (v)
oh = (slope * 1e9 / 100) * (0.1743 / 1e3) / 1000
c = 29.98  # speed of light in cm/ns
v = slope / c
```

$$h = vt \rightarrow v = \frac{h}{t}$$

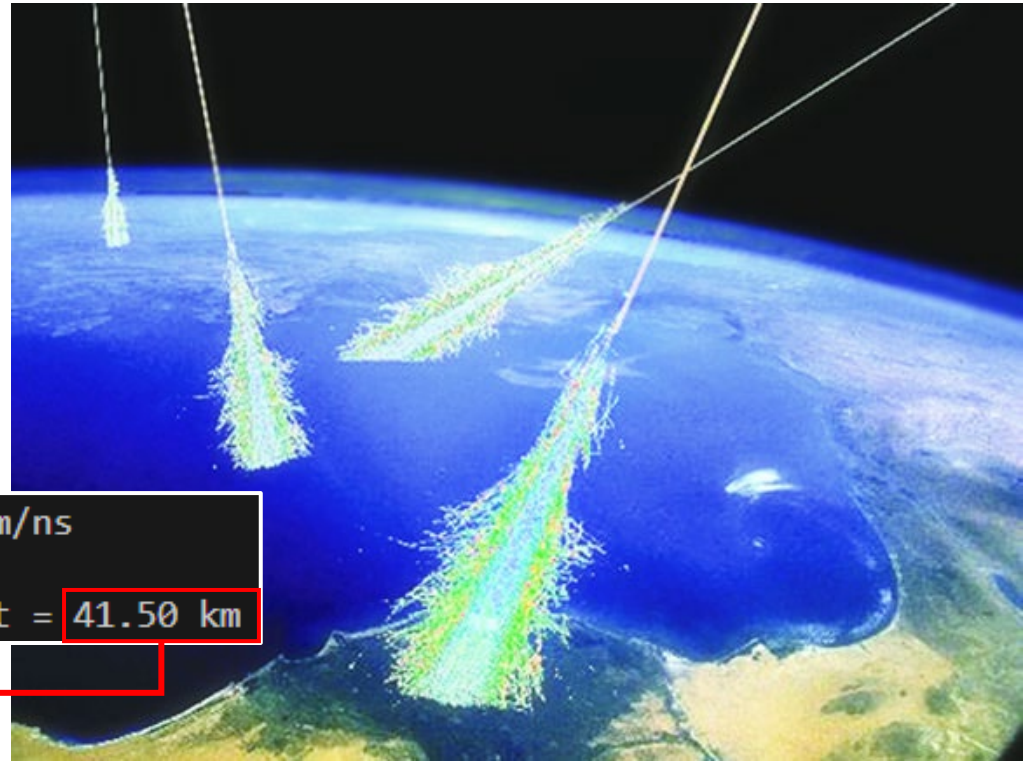
```
Slope = -23.8082 cm/ns
```

$$v = \frac{23.8082\ cm}{ns} \times \frac{10^9\ ns}{1\ s} \times \frac{1\ m}{100\ cm} \times \frac{1\ s}{299{,}709{,}000\ m} = \boxed{0.79\ c}$$

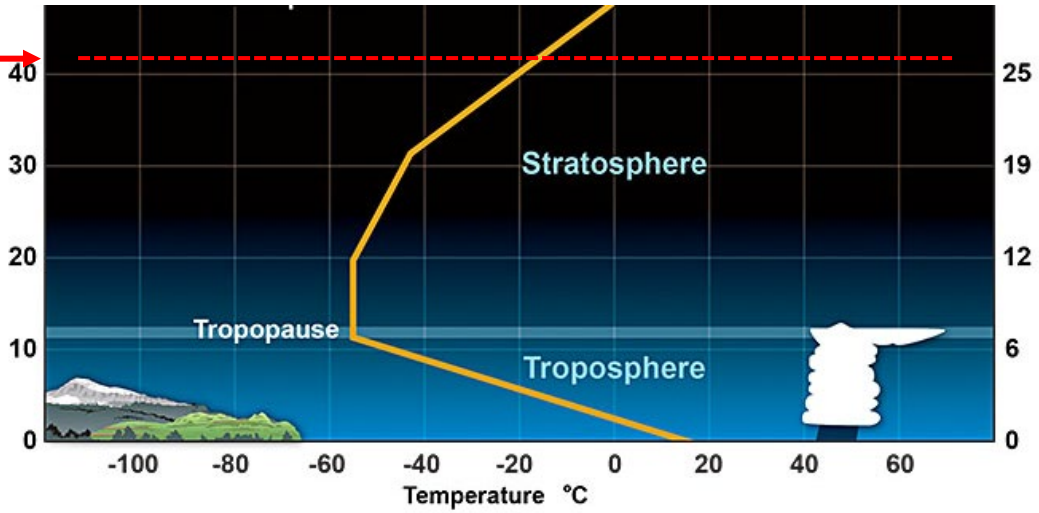
Given the lifetime of the particle

$$t = \frac{0.1743\ ms}{1} \times \frac{1\ s}{1000\ ms} = 0.0001743\ \text{s}$$

$$h = (238{,}082{,}000)\frac{m}{s} \times (0.0001743)\frac{s}{1} \times \frac{1\ km}{1000\ m} = \boxed{41.50\ km}$$

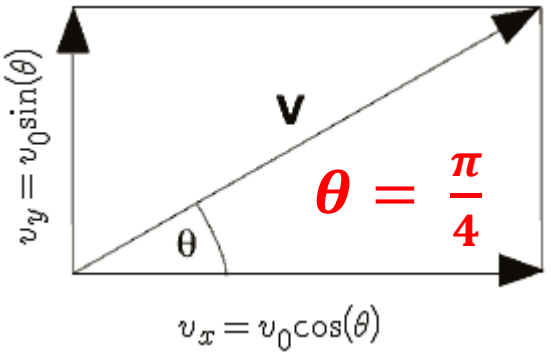


Slope = 23.8082 cm/ns
Velocity = 0.79 c
Origination Height = 41.50 km



Projectile Motion

Projectile Motion



v_0 = Initial velocity leaving the cannon

$$x = v_0 * t * \cos(\theta)$$

$$y = v_0 * t * \sin(\theta) - \frac{1}{2} g t^2$$

$$t = \frac{x}{v_0 * \cos(\theta)}$$

Given the Range = 30m, **what does v_0 need to be**?

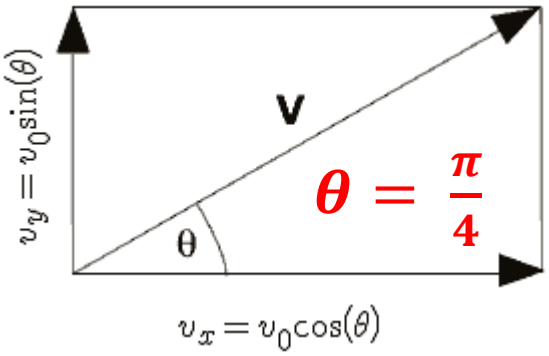
$$h = \frac{v_0^2 \sin^2(\theta)}{2g}$$

$$Range = \frac{4h}{\tan(\theta)}$$

$$y = \tan(\theta) * x - \frac{g}{2 * v_0^2 * cos^2(\theta)} * x^2$$

This is the **equation of motion** that allows us to **plot y as x increases** from launch point to trampoline

Projectile Motion



v_0 = Initial velocity leaving the cannon

$$x = v_0 * t * \cos(\theta)$$

$$y = v_0 * t * \sin(\theta) - \frac{1}{2} g t^2$$

$$t = \frac{x}{v_0 * \cos(\theta)}$$

Given the Range = 30m, **what does v_0 need to be**?

$$v_0 = \sqrt{\frac{Range * g}{\sin 2\theta}}$$

$$y = \tan(\theta) * x - \frac{g}{2 * v_0^2 * cos^2(\theta)} * x^2$$

This is the **equation of motion** that allows us to **plot y as x increases** from launch point to trampoline

Run projectile_motion.ipynb – **Cells 1...2**

Import needed packages/modules

```
[1]  # Cell 1
     import matplotlib.pyplot as plt
     import numpy as np
     from matplotlib.patches import Rectangle
```

① ←

Specify the simulation constants

1. The trampoline is 30m from the cannon
2. The cannon is at a fixed launch angle of 45°
3. The acceleration due to gravity is $9.81\ \frac{m}{s^2}$
4. The initial velocity v_0 is 15 m/s

② ←

```
[2]  # Cell 2
     rng = 30  # m
     theta = np.radians(45)  # 45 degree launch angle
     g = 9.81  # m/s^2
     v0 = 15 # m/s

     print(*[{k: v} for k, v in globals().items()
      if not k.startswith(('_','In','Out')) and not callable(v)], sep='\n')
```

③ ←

④ ←

```
{'plt': <module 'matplotlib.pyplot' from '/usr/local/lib/python3.10/dist-packages/matplotlib/pyplot.py'>}
{'np': <module 'numpy' from '/usr/local/lib/python3.10/dist-packages/numpy/__init__.py'>}
{'rng': 30}
{'theta': 0.7853981633974483}
{'g': 9.81}
{'v0': 15}
```

⑤ ←

Run projectile_motion.ipynb – **Cell 3**

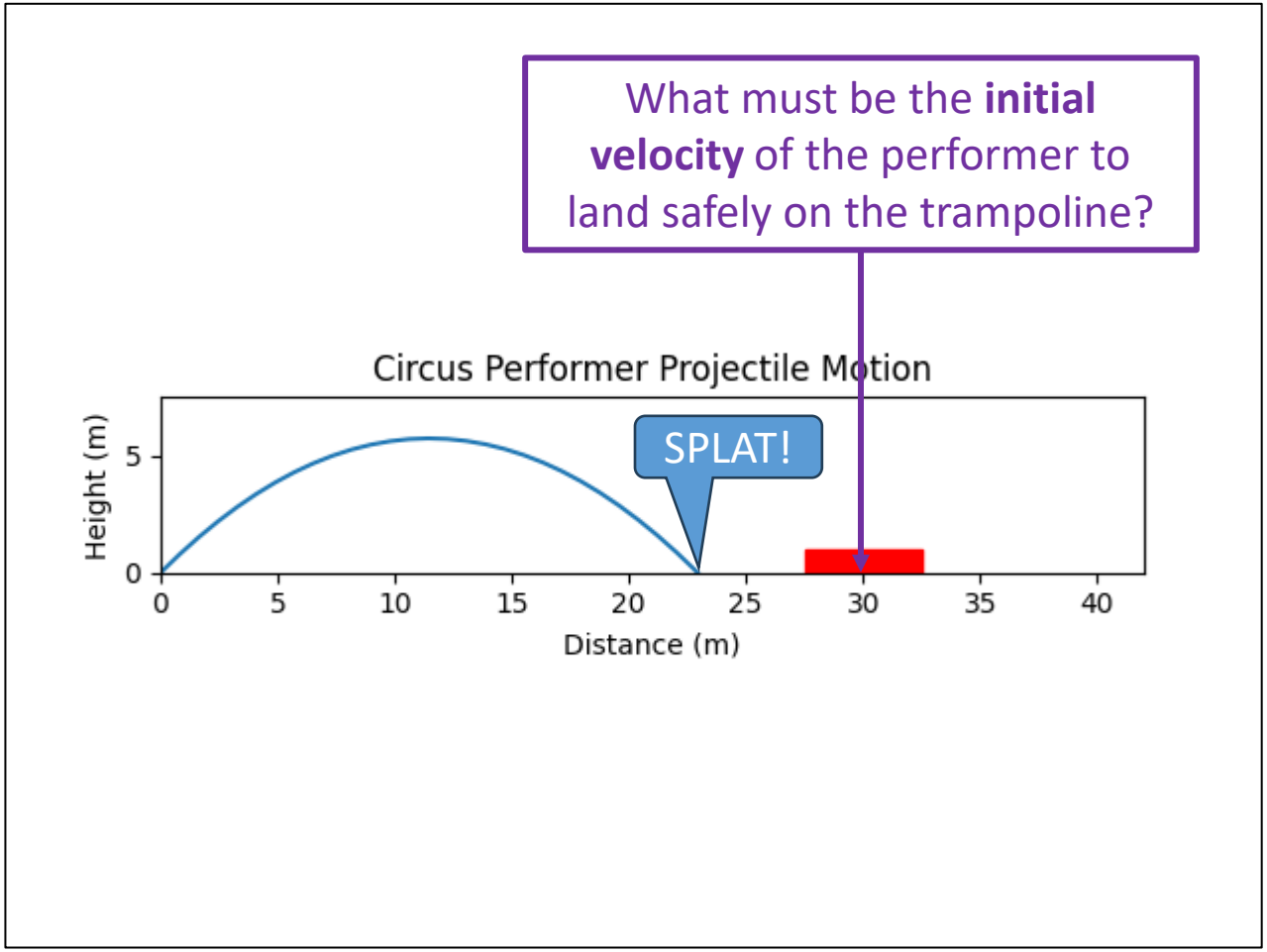
Create a function to calculate and plot the trajectory of the circus performer

- Create a linear space for the independent variable that spans $0 \leq x \leq 40$ ①
- $y = \tan(\theta) \times x - \frac{g}{2 \times v_0^2 \times \cos^2(\theta)} \times x^2$ ②

```
[3]  # Cell 3
     def plot_trajectory():
       x = np.linspace(0, 40)
       y = np.tan(theta) * x - (g / (2 * v0**2 * np.cos(theta) ** 2)) * x**2
       plt.plot(x, y)
       plt.title("Circus Performer Projectile Motion")
       plt.xlabel("Distance (m)")
       plt.ylabel("Height (m)")
       plt.xlim(left=0)
       plt.ylim(bottom=0)
       ax = plt.gca()
       ax.add_patch(Rectangle((27.5, 0), 5, 1, color="red"))
       ax.set_aspect("equal")
       plt.show()

     plot_trajectory()
```

③ ④ ⑤

Check projectile_motion.ipynb – **Cell 3**

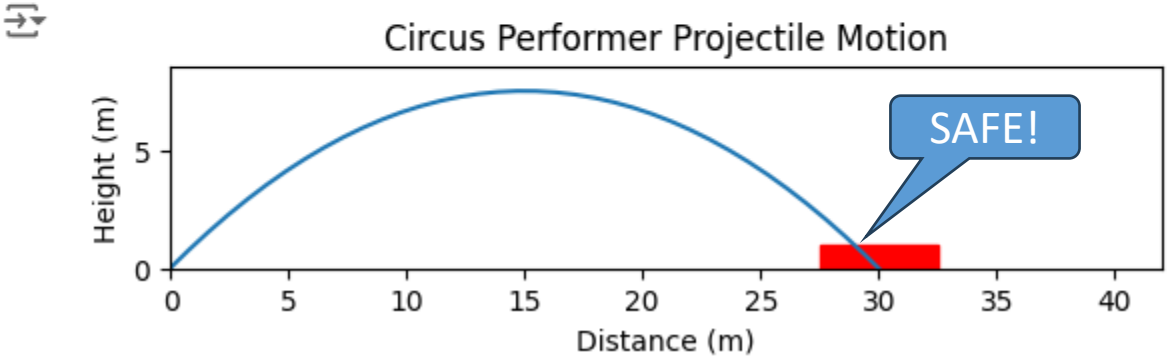


Run projectile_motion.ipynb – **Cell 4**

Set the correct initial launch velocity v_0 and recalculate the trajectory

$$v_0 = \sqrt{\frac{\text{Range} \times g}{\sin(2\theta)}}$$ ← ①

```
[4]  # Cell 4
     v0 = np.sqrt(rng * g / np.sin(2 * theta))
     plot_trajectory()
```
← ②



Session **03** – Topics

- Learn about Python's **dictionary** data structure and how to read data files in **CSV** and **JSON** formats
- Analyze chemical trends and anomalies across the **Periodic Table of Elements**
- Implement the **linear regression** formulas to find the **line of best fit** using the method of least squares created by Gauss
- Identify an *unknown element* using the **Ideal gas law**
- Determine the height and velocity of **cosmic ray** showers using Newtonian Kinematics
- Simulate the **trajectory** of a circus cannon performer to determine a safe initial launch velocity